VŠB - Technical University of Ostrava
Faculty of Mechanical Engineering
Department of Applied Mechanics

Diploma Thesis

Solution of Torsion of Beams with Non-Circular Cross-Sections with numerical methods

Řešení úloh kroucení prutů nekruhových průřezů numerickými metodami

Bc. Jan Blokeš

Supervisor: Ing. Alexandros Markopoulos Ph.D.

Field of Study: Applied Mechanics, 3901T003

Ostrava 2017

VŠB - Technical University of Ostrava
Faculty of Mechanical Engineering
Katedra aplikované mechaniky

Diploma Thesis Assignment

Student: **Bc. Jan Blokeš**

Study Programme: N2301 Mechanical Engineering

Study Branch: 3901T003 Applied Mechanics

Title: Solution of Torsion of Beams with Non-Circular Cross-Sections with numerical methods
Řešení úloh kroucení prutů nekruhových průřezů numerickými metodami

The thesis language: English

Description:

1. Torsion of beams with non-circular cross-sections (CS), an overview of the problem. Assumptions allowing the reduction of the problem dimension (from 3D to 2D dimension), the elastic membrane analogy.
2. Development of own software based on finite and/or boundary element method for solving of two dimensional problems.
3. Analysis of selected CS with the known analytical solution (circular, ellipsoid, square, rectangular CS etc.) in own library.
4. The comparison between own and commercial library on the general CS.

References:

1. S.P. Timoshenko, J.N. Goodier: Theory of Elasticity ISBN-13: 978-0070647206
2. Beer Gernot, Smith Ian, Duenser Christian: The Boundary Element Method with Programming, ISBN-13: 978-3211999004
3. Niels Saabye Ottosen, Hans Petersson: Introduction to the finite element method, ISBN 10: 0134738772
4. Roger Fenner: Boundary Element Methods for Engineers: Part I, ISBN: 978-87-403-0732-0

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Ing. Alexandros Markopoulos, Ph.D.**

Date of issue: 09.12.2016
Date of submission: 15.05.2017

doc. Ing. Radim Halama, Ph.D.
Head of Department

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA Fakulta strojní 3 OSTRAVA

doc. Ing. Ivo Hlavatý, Ph.D.
Dean

Declaration of the student

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used. I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

Píšť 14. 5. 2017 Student's signature .....

Prohlašuji, že

- jsem byl seznámen s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo.
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen „VŠB-TUO“) má právo nevýdělečně ke své vnitřní potřebě diplomovou práci užít (§ 35 odst. 3).
- souhlasím s tím, že diplomová práce bude v elektronické podobě uložena v Ústřední knihovně VŠB-TUO k nahlédnutí a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že údaje o kvalifikační práci budou zveřejněny v informačním systému VŠB-TUO.
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona.
- bylo sjednáno, že užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).
- beru na vědomí, že odevzdáním své práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, bez ohledu na výsledek její obhajoby

V Příšti dne 14. května 2017 ..... podpis

Jméno a příjmení autora práce: Jan Blokeš

Adresa trvalého pobytu autora práce: Májová 283/16, Příšť 747 18

Anotace diplomové práce

BLOKEŠ, J. *Řešení úloh kroucení prutů nekruhových průřezů numerickými metodami: diplomová práce*. Ostrava: VŠB - Technická univerzita Ostrava, Fakulta strojní, Katedra aplikované mechaniky, 2017, 168 s. Vedoucí práce: Markopoulos, A.

Diplomová práce popisuje téma volné kroucení prutů nekruhových průřezů. V první části je odvozená diferenciální rovnice, která je následně diskretizována metodami konečných a hraničních prvků. Tyto numerické metody byly ilustrovány na jednorozměrné i dvourozměrné úloze. Na dvourozměrné úloze je ukázána aplikace odvozené teorie a základy implementace vlastního software s názvem TorPy. Program umožňuje řešit kroucení libovolného profilu membránovou analogií metodou konečných prvků i metodou hraničních prvků. Výsledky jsou srovnány s analytickým řešením a složitější případy jsou porovnány s konečnoprvkovým softwarem Ansys.

Annotation of master thesis

BLOKEŠ, J. *Solution of Torsion of Beams with Non-Circular Cross-Sections with numerical methods: master thesis*. Ostrava: VŠB - Technical University of Ostrava, Faculty of Mechanical Engineering, Department of Applied Mechanics, 2017, 168 p. Head of thesis: Markopoulos, A.

This thesis deals with the torsion of beams with non-circular cross-section. The differential equation of the torsion is derived and discretized with finite element method and boundary element method. Examples of these methods are shown on one-dimensional and also two-dimensional cases. The derived theory is illustrated on two-dimensional examples. The author's software was developed to solve torsion with finite element method as well as boundary element method using membrane analogy. Results of the software TorPy are verified with available analytical solution and more complicated cases are compared with commercial software Ansys.

Contents

List of Abbreviations and Symbols **15**

Introduction **19**

1 Theory of torsion **21**
1.1 Introduction of torsion (General) . . . 21
1.2 Bars with circular cross-section . . . 22
1.3 Bars with non-circular cross-section . . . 25
1.3.1 Uniform torsion . . . 25
1.3.1a Prandtl's torsion theory . . . 25
1.3.1b Boundary conditions . . . 31
1.3.1c Determination of torque . . . 33
1.3.1d Comparison of membrane (Soap film) and torsion . . . 34
1.3.2 Non-uniform torsion . . . 34
1.4 The finite element method . . . 36
1.4.1 Simple triangular element . . . 39
1.4.2 Quadrilateral element . . . 43
1.4.2a Four node bilinear element = Lagrange's element . . . 43
1.4.2b Quadrilateral isoparametric element . . . 45
1.4.3 Solving system of linear equations . . . 47
1.4.3a Lagrange multipliers . . . 48
1.4.3b Conjugate gradient method . . . 49
1.5 The boundary element method . . . 51
1.5.1 Laplace equation . . . 52
1.5.1a Inverse formulation . . . 52
1.5.1b Fundamental solutions . . . 54
1.5.1c Boundary integral equation . . . 54
1.5.1d Discretisation of the boundary . . . 56
1.5.1e The Collocation Method . . . 59
1.5.1f Displacement in domain . . . 59
1.5.1g Calculation of element normals . . . 60
1.6 Comparison of FEM and BEM . . . 62

2 Numerical methods used for solving of torsion problems 65
2.1 Illustration on simple 1D problem . . . 65
2.1.1 FEM . . . 66
2.1.2 BEM . . . 68
2.2 Introduction of the problem . . . 70
2.3 FEM . . . 71
2.4 BEM . . . 74

3 Numerical analysis of chosen cross-sections using FEM and BEM 81
3.1 Introduction and analytical results for basic cross-sections . . . 81
3.1.1 Circle . . . 81
3.1.2 Rectangle . . . 81
3.1.3 Equilateral triangle . . . 82
3.1.4 Right-angled triangle . . . 83
3.1.5 Membrane analogy - numerical polar moment of inertia . . . 83
3.2 Application in author's software TorPy + author's mesh generator . . . 86
3.2.1 Author's Mesh generator . . . 86
3.2.1a Rectangle . . . 86
3.2.1b Right-angled triangle . . . 88
3.2.1c Equilateral triangle . . . 89
3.2.1d Circle . . . 90
3.2.2 Comparison of the cross-section properties . . . 91
3.2.2a Circle . . . 91
3.2.2b Square . . . 91
3.2.2c Rectangle . . . 91
3.2.2d Right-angled triangle . . . 92
3.2.2e Equilateral triangle . . . 92
3.2.3 Membrane analogy- numerical polar moment of inertia . . . 92
3.3 Application in author's software TorPy + external mesh generator . . . 93
3.3.1 Comparison with commercial software Ansys . . . 93
3.3.2 Modelling of the torsion in software Ansys . . . 94

4 Results 97
4.1 Application in author's software TorPy + own mesh generator . . . 97
4.1.1 Comparison of the cross-section properties . . . 97
4.1.1a Circular cross-section . . . 98
4.1.1b Square cross-section . . . 100
4.1.1c Rectangle cross-section . . . 101
4.1.1d Equilateral triangle cross-section . . . 103
4.1.1e Right-angled triangle cross-section . . . 105
4.1.2 Membrane analogy - numerical polar moment of inertia . . . 107
4.2 Application in author's software TorPy + external mesh generator . . . 111

5 Conclusions 113

Acknowledgment 115

Bibliography **121**

Table of contents for Appendix **129**

List of Abbreviations and Symbols

BEM	Boundary element method
CGM	Conjugate gradient method
CS	Cross-section
FE	Finite element
FEA	Finite element analysis
FEM	Finite element method
FDM	Finite difference method

a	mm	Side (Dimension)
$\mathbf{a}$	-	Vector of coordinates
A	mm^2	Area
$\mathbf{A}$	-	Stiffness-bordered matrix
b	mm	Dimension of rectangle
$\mathbf{b}$	-	Vector of coodinates
$\mathbf{B}$	-	Boolean matrix
$c(\xi)$	mm	Free term coefficient
$\mathbf{c}$	-	Arbitrary matrix
$\mathbf{C}$	-	**C**-matrix
d	mm	Diameter
d_1	mm	Outer diameter
d_2	mm	Inner diameter
$\mathbf{D}$	-	Constitutive matrix in plane stress
E	MPa	Young's modulus
$\mathbf{f}$	-	Load vector
$\mathbf{f_b}$	-	Boundary vector
$\mathbf{f_l}$	-	Load vector
F	N	Force
$\mathbf{g}$	-	Gradient
$\mathbf{g}_0$	-	Initial gradient
G	MPa	Shear modulus
$\mathbf{G}_F$	-	Derivative of shape function
$\mathbf{G}$	-	Matrix of coefficients

h	mm	Dimension of rectangle
$\mathbf{H}$	-	Matrix of coefficients
i	-	Counter, position of matrix component
$\mathbf{I}$	-	Unit matrix
I_p	mm^4	Polar moment of inertia
j	-	Counter, position of matrix component
$\mathbf{J}$	-	Jacobian matrix
k	N	Constant tension force per unit length
$\mathbf{K}$	-	Stiffness matrix
l	mm	Length
$\mathcal{L}$	-	Differential operator
L	mm	Length of a beam
$\mathbf{m}$	-	Tangential unit vector
m_x	-	Component of tangential unit vector in x-drirection
m_y	-	Component of tangential unit vector in y-direction
$\mathbf{M}$	-	Matrix of the degree of polynomial
n	-	Boundary normal
$\mathbf{n}$	-	Boundary normal vector
$\mathbf{N}$	-	Shape function
n_{nx}	-	Component of normal vector in x-direction
n_{nz}	-	Component of normal vector in y-direction
n_z	-	Component of normal vector in z-direction
p	MPa	Lateral pressure, parameter
p_{nx}	MPa	Traction vector component in x-direction
p_{nz}	MPa	Traction vector component in y-direction
p_{nz}	MPa	Traction vector component in z-direction
$\mathbf{p}$	-	Conjugate vector
$\mathbf{P}$	-	Preconditioning matrix
q	-	Derivative of unknown field
q^*	-	Fundamental solution of unknown field variable
$\bar{q}$	-	Known derivative of potential
$\hat{\mathbf{q}}$	-	Vector of derivatives of potential
r	mm	Radius
$\mathbf{r}$	-	Euclidean distance between load point and field point
s	mm	Domain of one element
T	Nm	Torque
u	mm	Displacement
u_x	mm	Displacement in x-direction
u_y	mm	Displacement in y-direction
u_z	mm	Displacement in z-direction
u^*	-	Fundamental solution of unknown field variable
$\bar{u}$	-	Known potential (displacement)
$\tilde{u}$	-	Potential of Laplace equation
$\mathbf{u}$	-	Vector of unknowns
$\hat{\mathbf{u}}$	-	Vector of potentials

v	-	Weight function
$\mathbf{v}$	-	Weight function vector
V	mm^3	Volume
V_ζ	mm^3	Volume under displacement ζ
V_ϕ	mm^3	Volume under displacement ϕ
x	mm	Coordinate
x'	mm	Position of point A' in x direction
x_0	mm	Coordinates of point 0 in x-direction
X	N/m^3	Body force in x-direction
y	mm	Coordinate
$\mathbf{x}$	-	Vector of unknown, Vector of coordinates of nodes
y'	mm	Position of point A' in y-direction
y_0	mm	Coordinates of point 0 in y-direction
$\mathbf{y}$	-	Vector of coordinates of nodes
Y	N/m^3	Body force in y-direction
z	mm	Coordinate
$\mathbf{z}$	-	Z-vector
$\mathbf{z}_0$	-	Initial z-vector
Z	N/m^3	Body force in z-direction

α	-	Parameter
$\boldsymbol{\alpha}$	-	Matrix of coefficients
β	-	Parameter
γ	-	Shear strain
γ_{xy}	-	Shear strain in xy plane
γ_{yz}	-	Shear strain in yz plane
γ_{xz}	-	Shear strain in xz plane
Γ	mm	Boundary of a domain
δ	-	Dirac's delta
Δ	-	Laplace operator
ε	mm	Euclidean distance between load point and field point
ε_x	-	Normal strain in x-direction
ε_y	-	Normal strain in y-direction
ε_z	-	Normal strain in z-direction
ζ	mm	Deflection of membrane
η	-	Coordinate of parent domain
ϑ	rad/m	Rate of twist (proportional twist)
κ	-	Coefficient
$\boldsymbol{\lambda}$	-	Lagrange multipliers
μ	-	Poisson's ratio
ξ	-	Load point,coordinate of parent domain
$\boldsymbol{\xi}$	-	Load point vector
π	-	Ludolph's number: pi, $\pi = 3.14159$
ρ	mm	Radius
σ_e	MPa	Equivalent stress
σ_x	MPa	Normal stress in x-direction
σ_y	MPa	Normal stress in y-direction
σ_z	MPa	Normal stress in z-direction
τ_{max}	MPa	Maximum shear stress
τ_{xy}	MPa	Shear stress in xy plane
τ_{yz}	MPa	Shear stress in yz plane
τ_{xz}	MPa	Shear stress in xz plane
$\phi(x, y)$	N/mm	Prandtl's stress function
φ	rad	Angle of twist
ψ	rad	Arbitrary angle
ω	-	Gauss point
Ω	mm^2	Domain
∇	-	Nabla operator
$\mathbf{0}$	-	Zero matrix

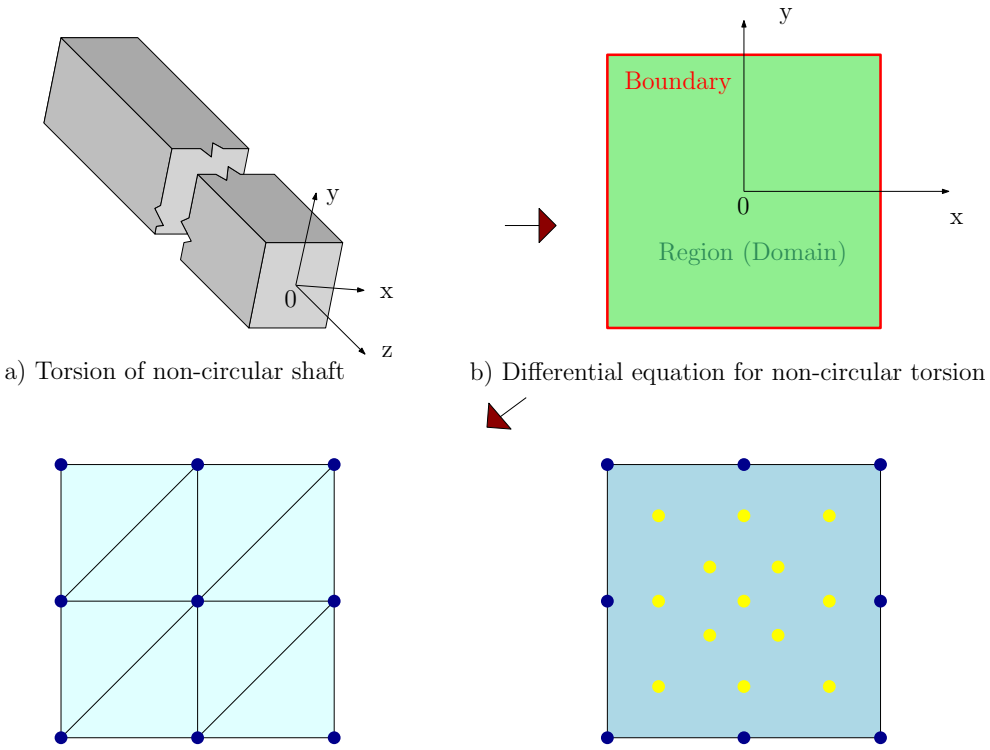
Introduction

Contemporary engineering requires to solve elaborate problems fast and with great amount of accuracy. More modern technologies are developed and implemented in engineering field. Experiments are still the most reliable technique of verification of a new project. However, so-called predictive engineering becomes more and more important. The predictive engineering uses modern computation methods, for example, the most used Finite Element Method.

The expansion of numerical methods has started with computers. The theory of these methods was derived before, but it was inefficient without computers. Both FEM and BEM, they are applicable in different fields of engineering, such as mechanics, heat transfer, electromagnetism and so on. The methods transform partial differential equations or equations to system of algebraic equations and the computer can solve them.

A torsion is one of the most common load type in mechanical engineering. If two dimensions are noticeably smaller than the third, the part can be considered as beam (bar). The beam is utilized as a substitution of three-dimensional parts and the calculation becomes strongly simpler. According to Saint-Venant torsion theory (see Chapter 1), three dimensional problem of torsion can be reduced into two-dimensional problem on condition, that warping is not restrained (1.37). The Saint-Venant theory results in differential equation, for mathematicians well-known as Poisson equation. The differential equation describes the behavior of real physical phenomenon. The exact solution of the differential equation exists only for few cross-sections, such as circular, annular, rectangle or elliptical. This type of solution is so-called analytical. However, for other cross-sections, the differential equation has to be solved numerically, for example the finite element method, a finite difference method or the boundary element method. These numerical methods approximates the exact solution of a geometry and a field variable (cf. Fig. 1)

The aim of thesis is to create a software for solving non-circular cross-section loaded by a torque or by a rate of twist. The software is implemented in programming language Python and it is able to solve this physical phenomenon either via FEM or BEM. The implementation is explained step by step and examples are included. The developed software is called TorPy, as an abbreviation of Torsion and Python.



a) Torsion of non-circular shaft

b) Differential equation for non-circular torsion

c) Finite element or boundary element mesh

Figure 1: Modelling procedure

The thesis follows in an older thesis [31]. The software is able to solve only linear problems. It is possible to transform FE mesh to BE mesh for arbitrary two-dimensional triangular mesh from commercial software MSC Patran or MSC Marc. The program can also automatically detect boundary conditions for this mesh and solve the problem as via FEM so via BEM. The cross-sections are verified with analytical solutions in case that they are known. If not, the solution is compared with commercial software ANSYS.

First chapter describes Saint - Venant theory of torsion, derivation of FE formulation as well as BE formulation and their comparison. Examples of mentioned methods in 1D and 2D are shown in next chapter. Third chapter is concerned with investigation of certain cross-sections. The analytical solution for basic cross-sections and the own mesh generator are introduced. The calculation of a numerical polar moment of inertia and comparison of software TorPy with commercial software Ansys is stated in Chapter 3 as well. Results are shown and commented in separated Chapter 4. The conclusion is written in the last chapter.

Chapter 1

Theory of torsion

1.1 Introduction of torsion (General)

For the entire theory in this thesis the following assumptions were taken:

- The material is homogeneous and isotropic
- Deformations are small and shear stress is less than yield strength
- Hooke's law is valid
- A load (a torque) is static
- The cross-section is closed and without holes (solid)
- A warping is free, i.e. $\sigma_z = 0$
- Transverse shape of the beam is constant, i.e. uniform cross-section

The theory is based on assumptions of linear elasticity. At first, it will be derived the theory of simple circular cross-section and then more general non-circular cross section.

1.2 Bars with circular cross-section

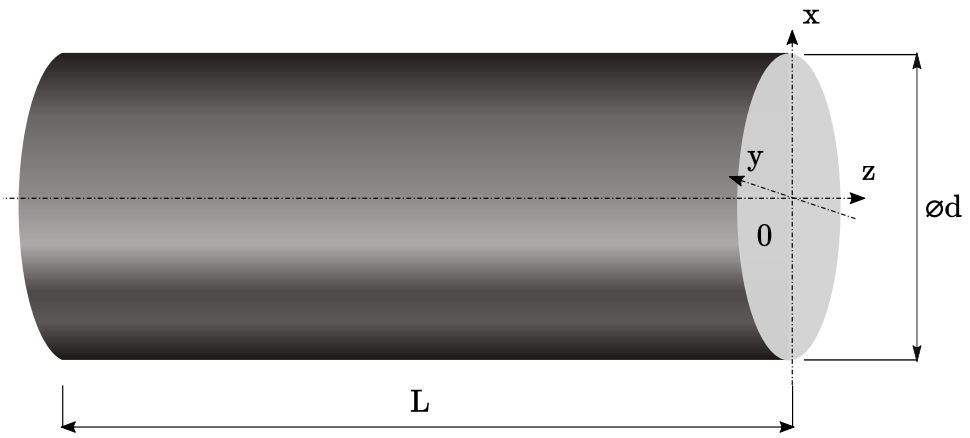


Figure 1.1: Bar with circular cross-section

The reader can imagine a simple cylinder, as shown in Fig. 1.1 , which may represent, for example, a shaft. The deformation of the bar is a consequence of applied torque (couple

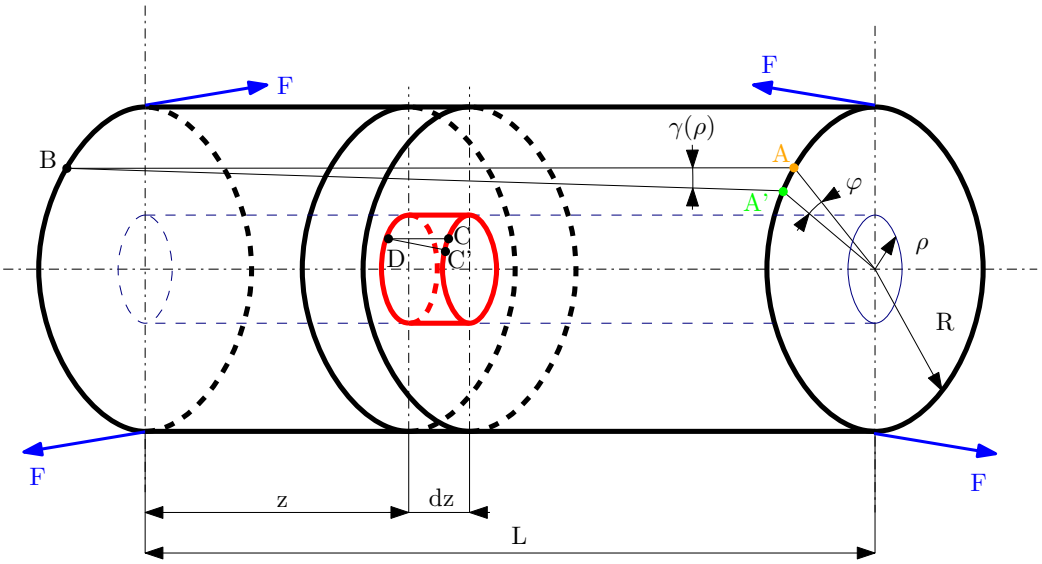


Figure 1.2: Deformed bar after application of a torque

of blue forces). The undeformed fiber AB is after deformation moved as fiber A'B (see Fig 1.2). The fiber AB has a shape of a straight line and the fiber A'B represents a helix. The transverse cross-sections are only turned around z-axis and they are not collapsed. In accordance with Fig. 1.3, a length of arc CC' (for small deformations) may be written as

$$\gamma(\rho) \cdot \mathrm{d}z = \mathrm{d}\varphi \cdot \rho \tag{1.1}$$

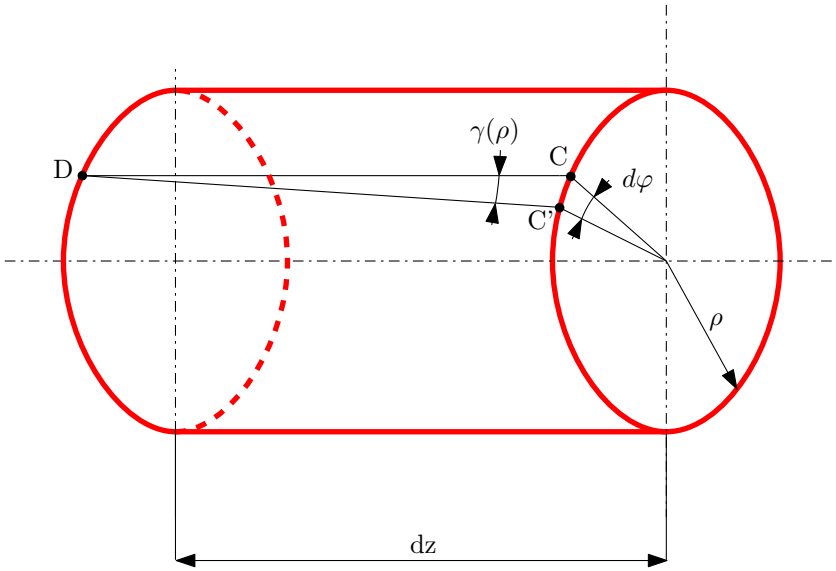


Figure 1.3: Zoom of the small element of the bar

where the shear strain γ is dependent on the radius ρ.

$$\gamma = \frac{\mathrm{d}\varphi}{\mathrm{d}z}\rho \ . \tag{1.2}$$

The fraction is so-called a rate of twist ϑ and it is a constant, i.e.

$$\vartheta = \frac{\mathrm{d}\varphi}{\mathrm{d}z} = \text{const.} \tag{1.3}$$

Hook's law for the shear stress is defined by

$$\tau = \gamma G \tag{1.4}$$

where G is a shear modulus which is calculated as

$$G = \frac{E}{2(1+\mu)} \ . \tag{1.5}$$

If Equations (1.2) and (1.4) are combined, it gives

$$\tau = \vartheta \rho G \ . \tag{1.6}$$

It is clear that the shear stress is proportional the radius ρ. It remains to express the shear stress with influence on the load, i.e. torque T. The torque is equal to a moment arm and a force. According to Fig. 1.4, the elementary force is equal to

$$\mathrm{d}F = \tau \, \mathrm{d}A \ . \tag{1.7}$$

The moment arm is the radius ρ, it means that the elementary torque is given by

$$\mathrm{d}T = \rho \, \mathrm{d}F = \rho\tau \, \mathrm{d}A \tag{1.8}$$

thus,

$$T = \iint_A \rho\tau \, \mathrm{d}A \ . \tag{1.9}$$

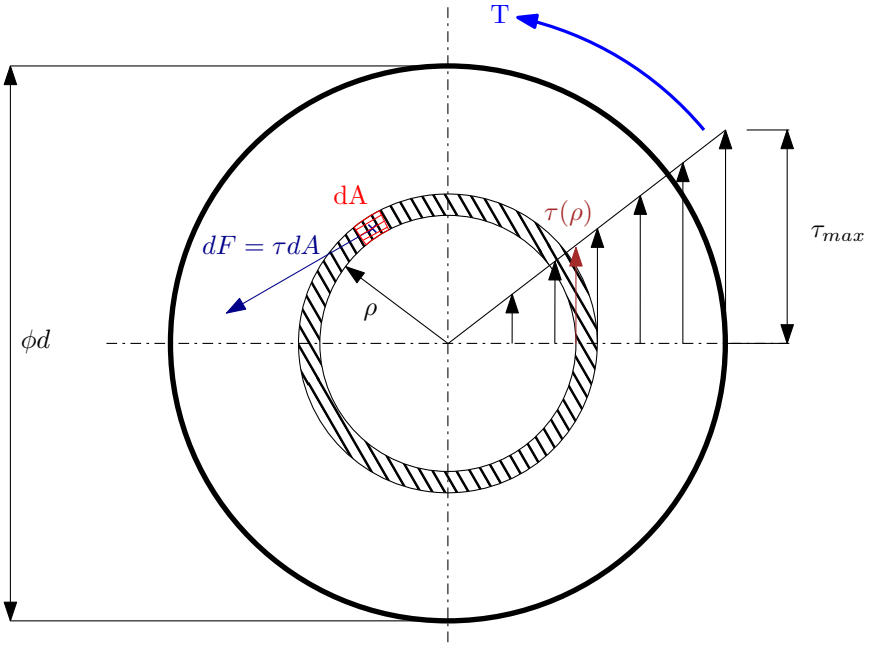


Figure 1.4: Circular cross-section with stress distribution

With (1.6),(1.9) implies

$$T = G\vartheta \iint_A \rho^2 \, \mathrm{d}A = G\vartheta I_p \tag{1.10}$$

where the rate of twist ϑ can be expressed as

$$\vartheta = \frac{T}{GI_p} \ . \tag{1.11}$$

Combining Eqs. (1.6) and (1.11) follows that

$$\tau = \frac{T}{I_p}\rho \tag{1.12}$$

where I_p is called a polar moment of inertia and for circular cross-section is defined as

$$I_p = \frac{\pi d^4}{32} \ . \tag{1.13}$$

The polar moment of inertia for annular cross-section gives

$$I_p = \frac{\pi(d_1^4 - d_2^4)}{32} \tag{1.14}$$

where d_1 is an outer diameter and d_2 denotes an inner diameter. Therefore, the maximum shear stress for any outer fiber, i.e. $\rho = \frac{d}{2}$, becomes

$$\tau_{max} = \frac{T}{\frac{\pi}{16} d^3} \tag{1.15}$$

and similarly, for the annular cross-section yields

$$\tau_{max} = \frac{T}{\frac{\pi}{16}\left(d_1^3 - \frac{d_2^4}{d_1}\right)} \ . \tag{1.16}$$

More information can be obtained for example from [14], [16], [22]

1.3 Bars with non-circular cross-section

1.3.1 Uniform torsion

Any longitudinal cross-section represents a helix after deformation. However, the transverse cross-sections do not remain flat and they are collapsed (see the cuboid in Fig.1.5). This type of transverse deformation is called a warping. If the uniform torsion is considered, the warping is the same for any transverse cross-section. If the end of the bar is welded, the warping is not the same for the transverse cross-sections and such a torsion is so-called non-uniform.

1.3.1a Prandtl's torsion theory

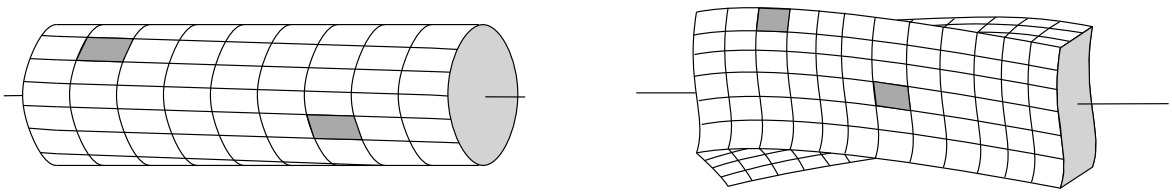


Figure 1.5: Deformation of circular bar and warping of a rectangular bar

The cross section can be arbitrary, it is assumed a rectangular (see Figure 1.6) for illustration purposes only. The shaft is loaded by torque T (i.e. twisting moment). It is chosen point A somewhere in cross-section. Due to torque, the bar is deformed, as shown in Fig. 1.7 and the point A transformes into point A'.

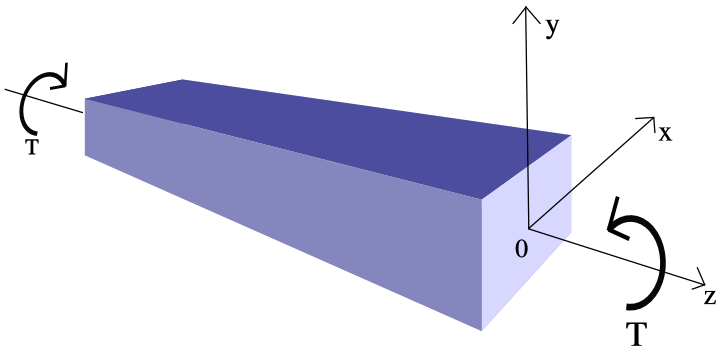


Figure 1.6: Torques acting on a non-circular bar

The illustration of a behavior of the cross-section in xy plane after torque application is shown in Fig. 1.7.

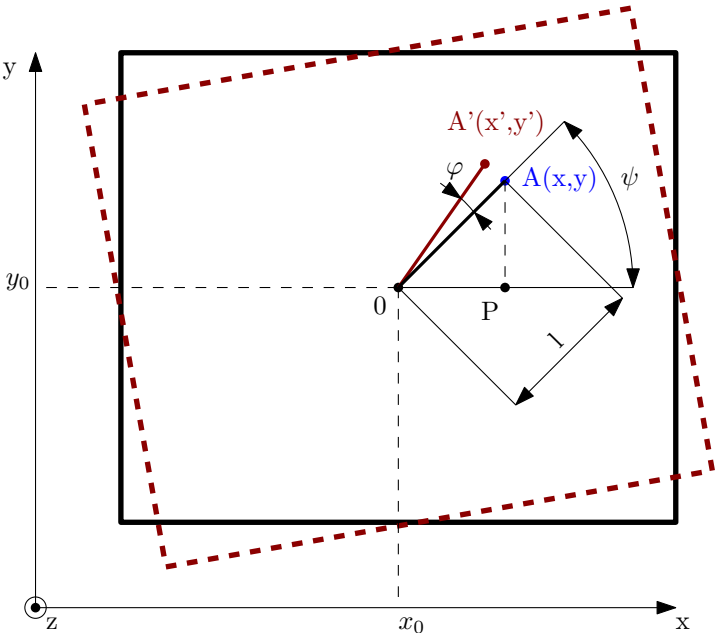


Figure 1.7: Cross-section before (black) and after deformation (dashed brown)

The position of point A in xy coordinate system according Fig. 1.7 is defined by

$$x = x_0 + l \cdot cos(\psi) \tag{1.17}$$

$$y = y_0 + l \cdot sin(\psi) \ . \tag{1.18}$$

Likewise, the position of point A' in the same coordinate system is defined by

$$x' = x_0 + l \cdot \cos(\psi + \varphi) \tag{1.19}$$

$$y' = y_0 + l \cdot \sin(\psi + \varphi) \ . \tag{1.20}$$

According to trigonometric formulas which can be found for instance in [24], the sine and cosine functions may be written as

$$\cos(\psi + \varphi) = \cos(\psi) \cdot \cos(\varphi) - \sin(\psi) \cdot \sin(\varphi) \tag{1.21}$$

$$\sin(\psi + \varphi) = \sin(\psi) \cdot \cos(\varphi) + \cos(\psi) \cdot \sin(\varphi) \ . \tag{1.22}$$

Thus, the Equations (1.19) and (1.20) with (1.21) and (1.22) become

$$x' = x_0 + l(\cos(\psi)\cos(\varphi) - \sin(\psi)\sin(\varphi)) \tag{1.23}$$

$$y' = y_0 + l(\sin(\psi)\cos(\varphi) + \cos(\psi)\sin(\varphi)) \ . \tag{1.24}$$

According to assumptions on page 21, the rotation angle φ has to be small. Therefore, the approximation may be defined as

$$cos(\varphi) \approx 1 \ \text{ and } \ sin(\varphi) \approx \varphi \tag{1.25}$$

and using (1.25) in (1.23) and (1.24) yields

$$x' = x_0 + l \cdot \cos(\psi) - l \cdot \varphi \cdot \sin(\psi) \tag{1.26}$$

$$y' = y_0 + l \cdot sin(\psi) + l \cdot \varphi \cos(\psi) \tag{1.27}$$

The Eq. (1.26) contains expression (1.17) as well as (1.18) is included in (1.27). This fact reduces to

$$x' = x - l \cdot \varphi \cdot \sin(\psi) \tag{1.28}$$

$$y' = y + l \cdot \varphi \cdot \cos(\psi) \ . \tag{1.29}$$

According to Figure 1.8, it is evident that

$$\sin\psi = \frac{y - y_0}{l} \tag{1.30}$$

$$\cos\psi = \frac{x - x_0}{l} \ . \tag{1.31}$$

The Equations (1.30) and (1.31) can be inserted into (1.28) and (1.29), i.e.

$$x' = x - \varphi \cdot (y - y_0) \tag{1.32}$$

$$y' = y + \varphi \cdot (x - x_0) \ . \tag{1.33}$$

Therefore, a displacement u_x in x-axis and and a displacement u_y in y-direction are determined as

$$u_x = x' - x = -\varphi \cdot (y - y_0) \tag{1.34}$$

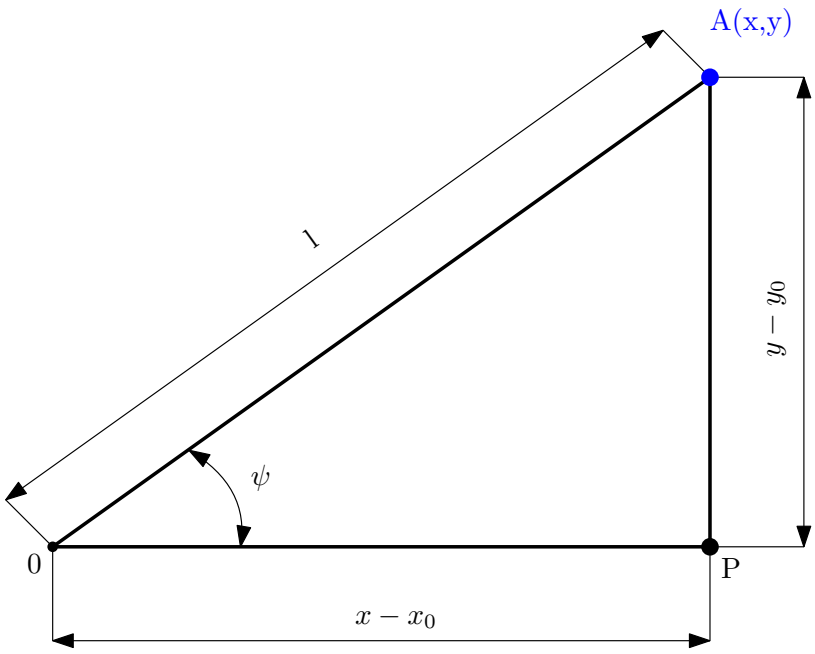


Figure 1.8: Zoom of the triangle 0PA

$$u_y = y' - y = \varphi \cdot (x - x_0) \ . \tag{1.35}$$

The angle φ is influenced by the coordinate z, it means the angle φ varies with the length of the beam L. It means that

$$\varphi = \varphi(z) \ . \tag{1.36}$$

The displacements in x- and y-directions were determined. As mentioned before, the displacement in z-direction is so-called warping and for uniform torsion depends on coordinates x and y. Thus, it is clear that

$$u_z = u_z(x, y) \ . \tag{1.37}$$

From the mathematical theory of elasticity [19] ,[23] or [26], there must be valid so-called strain - displacement relations. With (1.34), (1.35) and (1.37), normal strains ε_x,ε_y, ε_z and shear strains γ_{xy}, γ_{xz},γ_{yz} are defined as

$$\varepsilon_x = \frac{\partial u_x}{\partial x} = 0 \tag{1.38}$$

$$\varepsilon_y = \frac{\partial u_y}{\partial y} = 0 \tag{1.39}$$

$$\varepsilon_z = \frac{\partial u_z}{\partial z} = 0 \tag{1.40}$$

$$\gamma_{xy} = \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} = -\varphi + \varphi = 0 \tag{1.41}$$

$$\gamma_{xz} = \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} = -\frac{\mathrm{d}\varphi}{\mathrm{d}z}(y - y_0) + \frac{\partial u_z}{\partial x} \tag{1.42}$$

$$\gamma_{yz} = \frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} = \frac{\mathrm{d}\varphi}{\mathrm{d}z}(x - x_0) + \frac{\partial u_z}{\partial y} \ . \tag{1.43}$$

The non-zero strains are only γ_{xz} and γ_{yz}. The corresponding stresses from constitutive equations (Generalized Hook's Law) according to [23], [19] or [26] result in

$$\sigma_x = \sigma_y = \sigma_z = \tau_{xy} = 0 \tag{1.44}$$

$$\tau_{xz} = G\left(-\frac{\mathrm{d}\varphi}{\mathrm{d}z}(y - y_0) + \frac{\partial u_z}{\partial x}\right) \tag{1.45}$$

$$\tau_{yz} = G\left(\frac{\mathrm{d}\varphi}{\mathrm{d}z}(x - x_0) + \frac{\partial u_z}{\partial y}\right) \ . \tag{1.46}$$

Differential equations of equilibrium according to [23], [19] or [26] are defined by

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + X = 0 \tag{1.47}$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + Y = 0 \tag{1.48}$$

$$\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + Z = 0 \tag{1.49}$$

where X,Y and Z are body forces and they are neglected. With (1.44), (1.45) and (1.46), the equilibrium equations reduce to

$$\frac{\partial \tau_{zx}}{\partial z} = 0; \ \frac{\partial \tau_{zy}}{\partial z} = 0 \tag{1.50}$$

$$\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} = 0 \ . \tag{1.51}$$

The Eq. (1.3) for the rate of twist is valid for non-circular cross-section as well

$$\vartheta = \frac{\mathrm{d}\varphi}{\mathrm{d}z} = \text{const.} \tag{1.52}$$

The angle of twist for entire length of the bar is obtained by integration over the length of the beam L, i.e.

$$\int_0^{\varphi_L} \mathrm{d}\varphi = \vartheta \int_0^L \mathrm{d}z \ . \tag{1.53}$$

The resulting formula for the angle of twist is determined as

$$\varphi_L = \vartheta L \ . \tag{1.54}$$

Combining Eqs. (1.11) and (1.54), the angle of twist φ_L is proportional to torque, i.e.

$$\varphi_L = \frac{TL}{GI_p} \ . \tag{1.55}$$

Assume, that the shear stresses may be expressed as

$$\tau_{xz} = \frac{\partial \phi}{\partial y} \tag{1.56}$$

$$\tau_{yz} = -\frac{\partial \phi}{\partial x} \tag{1.57}$$

where $\phi = \phi(x, y)$ is called Prandtl's stress function. The stress function satisfies the equilibrium equations (1.50) and (1.51) as

$$\frac{\partial^2 \phi}{\partial x \partial y} - \frac{\partial^2 \phi}{\partial x \partial y} = 0 \ . \tag{1.58}$$

Combining (1.45), (1.46) with (1.56) and (1.57) and using (1.52) yields

$$\frac{\partial \phi}{\partial y} = G \left(-\vartheta (y - y_0) + \frac{\partial u_z}{\partial x} \right) \tag{1.59}$$

$$-\frac{\partial \phi}{\partial x} = G \left(\vartheta (x - x_0) + \frac{\partial u_z}{\partial y} \right) \ . \tag{1.60}$$

The partial derivatives of the stress function are given by

$$\frac{\partial}{\partial y} \left(\frac{\partial \phi}{G \partial y} \right) = -\vartheta + \frac{\partial^2 u_z}{\partial x \partial y} \tag{1.61}$$

$$-\frac{\partial}{\partial x} \left(\frac{\partial \phi}{G \partial x} \right) = \vartheta + \frac{\partial^2 u_z}{\partial x \partial y} \ . \tag{1.62}$$

With (1.61), (1.62) implies that

$$\frac{\partial}{\partial x} \left(\frac{\partial \phi}{G \partial x} \right) + \frac{\partial}{\partial y} \left(\frac{\partial \phi}{G \partial y} \right) + 2\vartheta = 0 \ . \tag{1.63}$$

and if the shear modulus G is considered as constant, so-called strong formulation is defined as

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = -2G\vartheta \ . \tag{1.64}$$

This type of differential equation is mathematically called Poisson's equation. A notation with Laplace (Laplacian) operator gives

$$\Delta \phi = -2G\vartheta \tag{1.65}$$

or the notation with Nabla operator results in

$$\nabla \cdot \nabla \phi = \nabla^2 \phi = -2G\vartheta \ . \tag{1.66}$$

The notation with divergence and gradient is determined as

$$\mathrm{div}(\mathbf{D} \nabla \phi) + 2\vartheta = 0 \tag{1.67}$$

where

$$\mathbf{D} = \begin{pmatrix} 1/G & 0 \\ 0 & 1/G \end{pmatrix} = \frac{1}{G}\mathbf{I} \tag{1.68}$$

is so-called constitutive matrix. The gradient of ϕ is defined as

$$\nabla\phi = \begin{pmatrix} \dfrac{\partial\phi}{\partial x} \\ \dfrac{\partial\phi}{\partial y} \end{pmatrix} \tag{1.69}$$

and the divergence given by

$$\mathrm{div}(\nabla\phi) = \frac{\partial}{\partial x}\frac{\partial\phi}{\partial x} + \frac{\partial}{\partial y}\frac{\partial\phi}{\partial y} \ . \tag{1.70}$$

The Eqs. (1.64) through (1.67) are identical. The solution of the equation exists only for simple geometries such as rectangle, equilateral triangle, etc. However, more complicated shape of the cross-section (for example in Fig. 3.15) must be solved numerically.

1.3.1b Boundary conditions

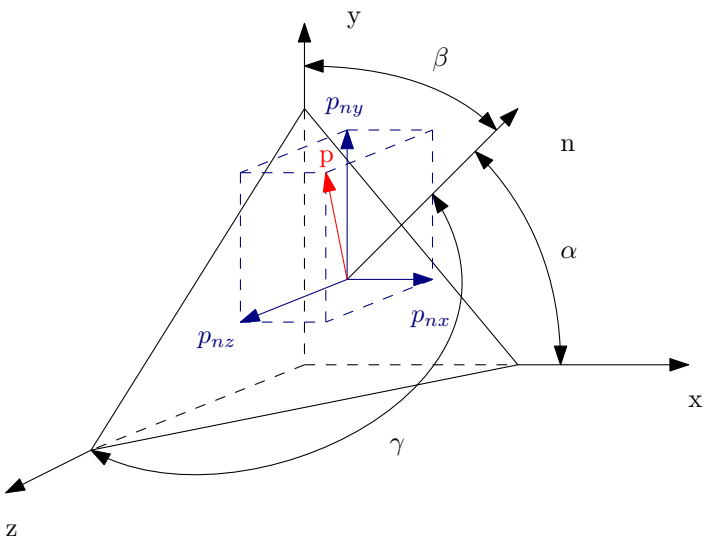


Figure 1.9: Traction vector on tetrahedron

From the surface of elementary (infinitesimal) tetrahedron, which is loaded by external forces, the equilibrium equations are obtained as follows

$$p_{nx} = \sigma_x n_x + \tau_{xy} n_y + \tau_{xz} n_z \tag{1.71}$$

$$p_{ny} = \tau_{yx} n_x + \sigma_y n_y + \tau_{yz} n_z \tag{1.72}$$

$$p_{nz} = \tau_{zx} n_x + \tau_{zy} n_y + \sigma_z n_z \tag{1.73}$$

where n_x,n_y and n_z are components of normal vector and p_{nx},p_{ny} and p_{nz} denotes forces on external surface. The derivation is in [25] or [23]. From Equation (1.44) implies that

$$p_{nz} = \tau_{zx} n_x + \tau_{zy} n_y \ . \tag{1.74}$$

Surfaces of the bar do not contain any load, i.e.

$$p_{nx} = p_{ny} = p_{nz} = 0 \ . \tag{1.75}$$

Thus, the equation (1.74) becomes

$$\tau_{zx} n_x + \tau_{zy} n_y = 0 \ . \tag{1.76}$$

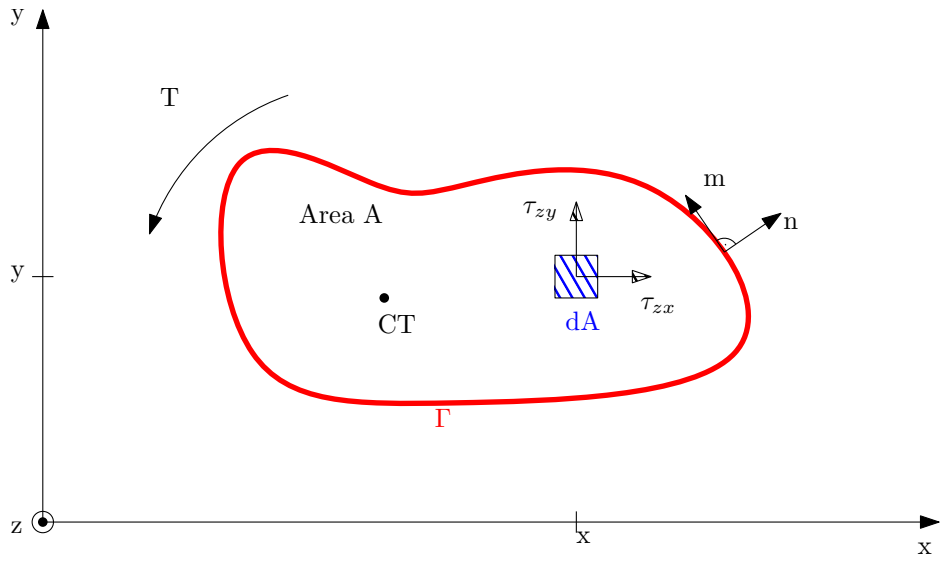


Figure 1.10: Establishment of boundary conditions and torque

According to Figure 1.10, let **n** represents unit vector, directed outwards from the cross-section and orthogonal to boundary. The tangential unit vector denotes vector **m**. A relation between these vectors is given by

$$\mathbf{n}^\top \mathbf{m} = 0 \quad \text{or} \quad n_x m_x + n_y m_y = 0 \tag{1.77}$$

where

$$\mathbf{n} = \begin{pmatrix} n_x \\ n_y \end{pmatrix} ; \ \mathbf{m} = \begin{pmatrix} m_x \\ m_y \end{pmatrix} \ . \tag{1.78}$$

Due to orthogonality of vectors **m** and **n**, it is valid that

$$\begin{pmatrix} n_x \\ n_y \end{pmatrix} = \begin{pmatrix} -m_y \\ m_x \end{pmatrix} \ . \tag{1.79}$$

From (1.76), (1.56),(1.57) and (1.79), it follows that

$$\frac{\partial \phi}{\partial x} m_x + \frac{\partial \phi}{\partial y} m_y = 0 \tag{1.80}$$

or

$$(\nabla \phi)^\top \mathbf{m} = 0 \ . \tag{1.81}$$

It is proved, that the formula (1.82) is valid [25]

$$\frac{\mathrm{d}\phi}{\mathrm{d}m} = (\nabla \phi)^\top \mathbf{m} \ . \tag{1.82}$$

Now, it is evident that

$$\frac{d\phi}{dm} = 0 \tag{1.83}$$

where Eq. (1.83) shows that ϕ is unvarying along the boundary Γ. It is convenient to set

$$\phi = 0 \text{ on boundary } \Gamma \ . \tag{1.84}$$

The boundary condition (1.84) is valid only for a solid cross-section without holes. In consideration of the assumptions from Page 21, it is sufficient.

1.3.1c Determination of torque

The load in differential equation (1.64) is applied via the rate of twist ϑ. However, the most often known load is a torque. The task is to connect the torque and the rate of twist somehow. In order to determine torque, the small element in arbitrary point with coordinates x and y is used (see Figure 1.10). The static equilibrium must be valid. It is taken a moment about the origin of the coordinate system as

$$T = \iint_A (\tau_{zy} x - \tau_{zx} y) \mathrm{d}A \ . \tag{1.85}$$

Insertion of (1.56) and (1.57) into (1.85) yields

$$T = -\iint_A \left(\frac{\partial \phi}{\partial x} x + \frac{\partial \phi}{\partial y} y \right) \mathrm{d}A \ . \tag{1.86}$$

A product rule is defined by

$$\frac{\partial}{\partial x}(\phi x) = \frac{\partial \phi}{\partial x} x + \phi \tag{1.87}$$

$$\frac{\partial}{\partial y}(\phi y) = \frac{\partial \phi}{\partial y} y + \phi \ . \tag{1.88}$$

Combining Eqs. (1.86) through (1.88) follows that

$$T = -\iint_A \left(\frac{\partial}{\partial x}(\phi x) + \frac{\partial}{\partial y}(\phi y) \right) \mathrm{d}A + 2 \iint_A \phi \, \mathrm{d}A \tag{1.89}$$

Using Green-Gauss theorem (see Appendix B), the first term in (1.89) gives

$$\iint_A \left(\frac{\partial}{\partial x}(\phi x) + \frac{\partial}{\partial y}(\phi y) \right) \mathrm{d}A = \int_\Gamma (\phi x \, \mathrm{d}y - \phi y \, \mathrm{d}x) \, \mathrm{d}\Gamma = 0 \tag{1.90}$$

due to (1.84). Finally, it follows that the torque results in

$$T = 2 \iint_A \phi(x, y) \, \mathrm{d}A \ . \tag{1.91}$$

Notice that Eq. (1.91) represents a volume under ϕ (the shape is called a spherical cap or a membrane), i.e.

$$V_\phi = \iint_A \phi(x, y) \, \mathrm{d}x\mathrm{d}y \ . \tag{1.92}$$

The relations (1.91) and (1.92) leads to

$$T = 2V_\phi \ . \tag{1.93}$$

The physical interpretation means that the torque is equal to double volume under the spherical cap. The spherical caps are shown in Section 4.1.2. As mentioned previously, usually the load is given by torque, however the differential equation contains the rate of twist. Due to assumption of linear behavior, the rate of twist is expressed as function of the torque

$$\vartheta = \frac{\vartheta_1}{T_1} T \tag{1.94}$$

where ϑ_1 is arbitrary value of rate of twist and T_1 is its corresponding torque. The solution ϕ is proportional to the torque T as well as the stresses τ_{xz} and τ_{yz}. The theory is taken from [19], [25] and [23]

1.3.1d Comparison of membrane (Soap film) and torsion

The derivation of the differential equation of the membrane can be found in [17] or [25]. The differential equation is defined as

$$\frac{\partial^2 \zeta}{\partial x^2} + \frac{\partial^2 \zeta}{\partial y^2} = -\frac{p}{k} \tag{1.95}$$

where k is a constant tension force per unit length, p denotes a lateral pressure and ζ represents a deflection of membrane (see Figure 1.11). Note that the differential equation of the membrane is very similar to Equation (1.64). The properties of both differential equations are compared in Tab. 1.1.

1.3.2 Non-uniform torsion

Due to constraints, the normal stress is nonzero, i.e.

$$\sigma_z \neq 0 \tag{1.106}$$

and the warping is not free. The theory is more difficult and it is usually used 3D modeling via numerical methods, such as FEM, BEM or FDM. The reader can find detailed description for example in [29] or [9].

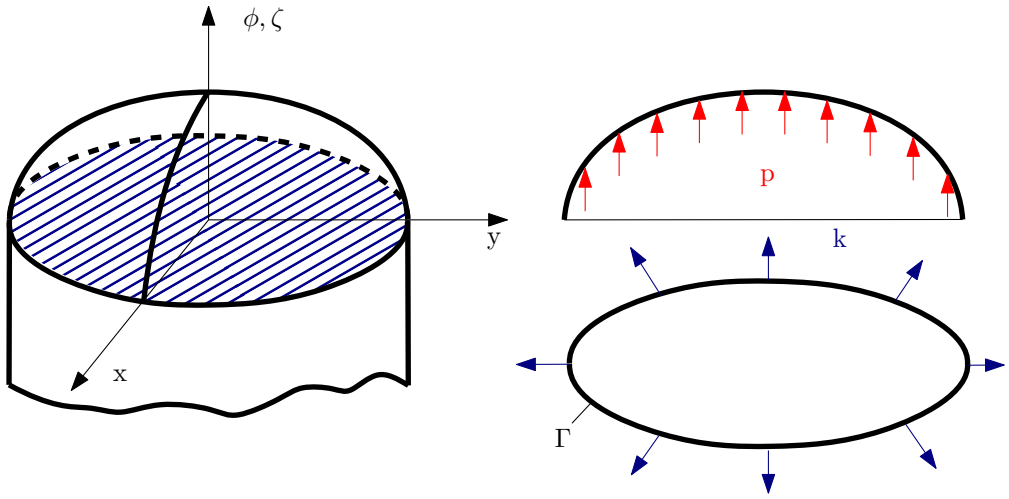


Figure 1.11: Membrane ζ and function ϕ

Membrane	**Torsion**
The equation of membrane: $$\frac{\partial^2 \zeta}{\partial x^2} + \frac{\partial^2 \zeta}{\partial y^2} = -\frac{p}{k} \quad (1.96)$$	The equation of torsion: $$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = -2G\vartheta \quad (1.97)$$
The displacement of the membrane: $$\zeta = \zeta(x, y) \quad (1.98)$$	The stress function: $$\phi = \phi(x, y) \quad (1.99)$$
Boundary conditions: $$\zeta = 0 \text{ on boundary } \Gamma \quad (1.100)$$	Boundary conditions: $$\phi = 0 \text{ on boundary } \Gamma \quad (1.101)$$
The volume under ζ: $$\iint_A \zeta(x, y) \mathrm{d}A = V_\zeta \quad (1.102)$$	The volume under ϕ = half torque: $$\frac{T}{2} = \iint_A \phi(x, y) \mathrm{d}A = V_\phi \quad (1.103)$$
The right-hand side: $$2G\vartheta \quad (1.104)$$	The right-hand side: $$\frac{p}{k} \quad (1.105)$$

Table 1.1: Membrane analogy

1.4 The finite element method

The derivation of the FE formulation is possible via Minimum Potential Energy or Weighted Residual Methods (from reference[25]). In this thesis, the weighted residuals are used. The reader can find more information about FEM in [2],[12],[6] or [7].

The equations (1.64) through (1.67) represent the strong formulation and its corresponding boundary condition (1.84). The derivation is similar to two-dimensional heat flow or diffusion due to similar differential equation.

The first step is multiplying the equation (1.67) by arbitrary weight function and integration over area A (for 3D problem over volume V), i.e.

$$\iint_A \left[v \operatorname{div}(\mathbf{D}\nabla\phi) + 2v\vartheta\right] \mathrm{d}A = 0 \ . \tag{1.107}$$

Then, the Green-Gauss theorem is applied to the first term of Eq. (1.107)

$$\iint_A v \operatorname{div}(\mathbf{D}\nabla\phi)\,\mathrm{d}A = \oint_\Gamma v(\mathbf{D}\nabla\phi)^\top \mathbf{n}\,\mathrm{d}\Gamma - \iint_A (\nabla v)^\top (\mathbf{D}\nabla\phi)\,\mathrm{d}A \ . \tag{1.108}$$

Combining (1.107) and (1.108) follows so-called weak formulation

$$\iint_A (\nabla v)^\top \mathbf{D}\nabla\phi\,\mathrm{d}A = \oint_\Gamma v(\mathbf{D}\nabla\phi)^\top \mathbf{n}\,\mathrm{d}\Gamma + \iint_A 2v\vartheta\,\mathrm{d}A \ . \tag{1.109}$$

It is possible to investigate the boundary term more detailed. From (1.68) it is obvious that

$$\mathbf{D}\nabla\phi = \frac{1}{G}\nabla\phi \ . \tag{1.110}$$

The equation can be multiplied by normal vector $\mathbf{n}$, i.e.

$$(\mathbf{D}\nabla\phi)^\top \mathbf{n} = \frac{1}{G}(\nabla\phi)^\top \mathbf{n} \ . \tag{1.111}$$

According to Eq. (1.82), it follows that

$$\frac{\mathrm{d}\phi}{\mathrm{d}n} = (\nabla\phi)^\top \mathbf{n} \ . \tag{1.112}$$

Insertion of (1.112) in (1.111) yields

$$(\mathbf{D}\nabla\phi)^\top \mathbf{n} = \frac{1}{G}\frac{\mathrm{d}\phi}{\mathrm{d}n} \ . \tag{1.113}$$

By inserting the formula (1.113) into (1.109) implies that

$$\iint_A (\nabla v)^\top \mathbf{D}\nabla\phi\,\mathrm{d}A = \int_{\Gamma_g} v\frac{\mathrm{d}\phi}{G\mathrm{d}n}d\Gamma + \int_{\Gamma_h} v\frac{\mathrm{d}\phi}{G\mathrm{d}n}\mathrm{d}\Gamma + 2\iint_A v\vartheta\,\mathrm{d}A \ . \tag{1.114}$$

The boundary Γ of the area A of the cross-section can be divided, as shown in Fig. 1.12, i.e.

$$\phi = 0 \text{ along } \Gamma_g \tag{1.115}$$

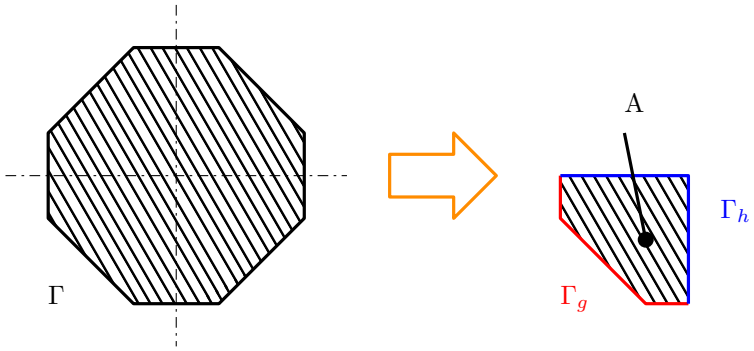


Figure 1.12: The boundary

$$\frac{1}{G}\frac{\mathrm{d}\phi}{\mathrm{d}n} \text{ along } \Gamma_h \ . \tag{1.116}$$

The blue boundary may represent a symmetry of the cross-section, particularly $\mathrm{d}\phi/\mathrm{d}n = 0$ and it will be used in programming later. The approximation function for the unknown field ϕ is determined by

$$\phi = \mathbf{N}\mathbf{u} \tag{1.117}$$

where

$$\mathbf{N} = \begin{pmatrix} N_1 & N_2 & \dots & N_n \end{pmatrix}; \ \mathbf{u} = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} \ . \tag{1.118}$$

The matrix $\mathbf{N}$ is called a global shape function matrix and the vector $\mathbf{u}$ includes the displacement ϕ at the nodal points. The gradient of ϕ is needed according to (1.114), thus

$$\nabla\phi = \mathbf{G}_F\mathbf{u} \tag{1.119}$$

where

$$\mathbf{G}_F = \begin{pmatrix} \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_2}{\partial x} & \dots & \dfrac{\partial N_n}{\partial x} \\ \dfrac{\partial N_1}{\partial y} & \dfrac{\partial N_2}{\partial y} & \dots & \dfrac{\partial N_n}{\partial y} \end{pmatrix} \tag{1.120}$$

and vector $\mathbf{u}$ does not depend on coordinates. It remains to choose the weight (test) function. According to Galerkin method, the weight function is the same as basis (trial) function, i.e.

$$v = \mathbf{N}\mathbf{c} \ . \tag{1.121}$$

The matrix $\mathbf{c}$ is arbitrary, since v is also arbitrary. An important property of the function is $v = v^\top$, because v is a scalar. It follows that

$$v^\top = v = \mathbf{c}^\top\mathbf{N}^\top \ . \tag{1.122}$$

From (1.121), it is obtained that

$$\nabla v = \mathbf{G}_F\mathbf{c} \tag{1.123}$$

and with (1.122) yields

$$\nabla v^\top = \mathbf{c}^\top \mathbf{G}_\mathbf{F}^\top \ . \tag{1.124}$$

Inserting (1.119), (1.122) and (1.124) into (1.114) implies that

$$\mathbf{c}^\top \left[\left(\iint_A (\mathbf{G}_F^\top \mathbf{D} \mathbf{G}_F \, \mathrm{d}A \right) \mathbf{u} - \int_{\Gamma_g} \mathbf{N}^\top \frac{\mathrm{d}\phi}{G \mathrm{d}n} \, \mathrm{d}\Gamma - \int_{\Gamma_h} \mathbf{N}^\top \frac{\mathrm{d}\phi}{G \mathrm{d}n} \, \mathrm{d}\Gamma - 2 \iint_A \mathbf{N}^\top \vartheta \, \mathrm{d}A \right] = 0 \ . \tag{1.125}$$

The matrix $\mathbf{c}$ is arbitrary and it may vanish so that

$$\left(\iint_A (\mathbf{G}_F^\top \mathbf{D} \mathbf{G} \, \mathrm{d}A \right) \mathbf{u} = \int_{\Gamma_g} \mathbf{N}^\top \frac{\mathrm{d}\phi}{G \mathrm{d}n} d\Gamma + \int_{\Gamma_h} \mathbf{N}^\top \frac{\mathrm{d}\phi}{G \mathrm{d}n} \, \mathrm{d}\Gamma + 2 \iint_A \mathbf{N}^\top \vartheta \, \mathrm{d}A \ . \tag{1.126}$$

Equation (1.126) may be written as

$$\mathbf{K}\mathbf{u} = \mathbf{f_b} + \mathbf{f_l} \tag{1.127}$$

where $\mathbf{K}$ is the stiffness matrix

$$\mathbf{K} = \iint_A \mathbf{G}_\mathbf{F}^\top \mathbf{D} \mathbf{G}_F \, \mathrm{d}A \tag{1.128}$$

$\mathbf{f}_b$ represents the boundary vector

$$\mathbf{f_b} = \int_{\Gamma_h} \mathbf{N}^\top \frac{\mathrm{d}\phi}{G \mathrm{d}n} \, \mathrm{d}\Gamma + \int_{\Gamma_g} \mathbf{N}^\top \frac{\mathrm{d}\phi}{G \mathrm{d}n} \, \mathrm{d}\Gamma \tag{1.129}$$

and $\mathbf{f}_l$ is called the load vector

$$\mathbf{f_l} = 2\vartheta \iint_A \mathbf{N}^\top \, \mathrm{d}A \ . \tag{1.130}$$

The sum of $\mathbf{f}_l$ and $\mathbf{f}_b$ is so-called the force vector $\mathbf{f}$. Therefore, the right-hand side may be written as

$$\mathbf{f} = \mathbf{f_l} + \mathbf{f_b} \ . \tag{1.131}$$

It is concluded with standard formula of the static finite element method

$$\mathbf{K}\mathbf{u} = \mathbf{f} \ . \tag{1.132}$$

The torque is derived on page 34 as (1.91). Using the approximation (1.117), it follows that

$$T = 2 \left(\iint_A \mathbf{N} \mathrm{d}A \right) \mathbf{u} \ . \tag{1.133}$$

However, the area A will be discretized by finite elements and it is more convenient to write (1.133) as

$$T_1 = 2 \sum_{i=0}^{n_e} \left[\left(\iint_{A^e} \mathbf{N}^e \mathrm{d}A \right) \mathbf{u}^e \right] \ . \tag{1.134}$$

The last object is the investigation of the shear stresses. If the potential (displacement) is known, then the gradient within each element can be computed as

$$\nabla \phi = \mathbf{G}_F^e \mathbf{u}^e \tag{1.135}$$

where the gradient $\nabla\phi$ corresponds (1.69). The stresses are defined in accordance with (1.56) and (1.57) as

$$\tau_{xz} = \frac{\partial \phi}{\partial y} \tag{1.136}$$

$$\tau_{yz} = -\frac{\partial \phi}{\partial x} \ . \tag{1.137}$$

Finally, the maximum stress within each element is calculated as

$$\tau_{max} = \sqrt{\tau_{xz}^2 + \tau_{yz}^2} \ . \tag{1.138}$$

The derivation is taken mostly from [25].

1.4.1 Simple triangular element

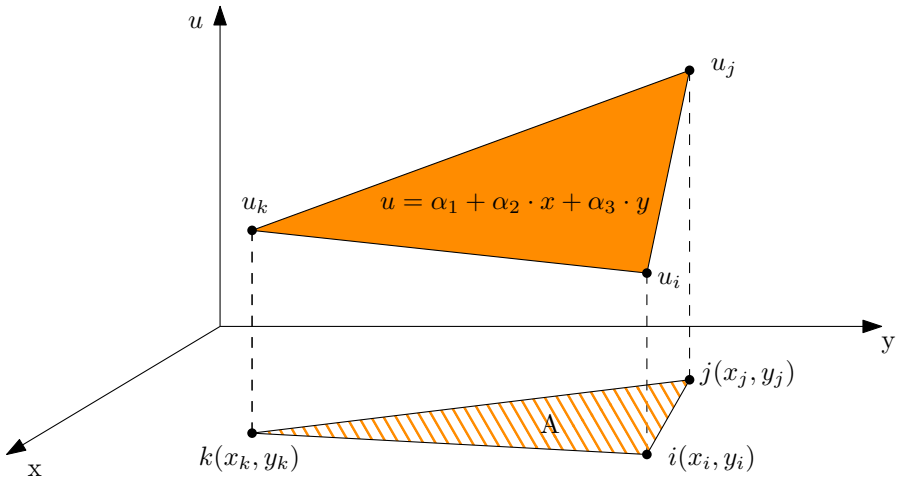


Figure 1.13: Triangular element

The simplest possible two-dimensional element is a triangle and its approximate function represents the equation of a plane (see Fig. 1.13). The matrix notation of the equation corresponds

$$u = \mathbf{M}\boldsymbol{\alpha} \tag{1.139}$$

where

$$\mathbf{M} = \begin{pmatrix} 1 & x & y \end{pmatrix} \tag{1.140}$$

and

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} \ . \tag{1.141}$$

The matrix $\mathbf{M}$ determines a degree of polynomial, so-called basis function. The matrix $\boldsymbol{\alpha}$ contains unknown coefficients.

The known coordinates of nodes can be inserted into Eq. (1.139). It gives

$$\begin{pmatrix} u_i \\ u_j \\ u_k \end{pmatrix} = \begin{pmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} \quad . \tag{1.142}$$

where

$$\mathbf{u}^e = \begin{pmatrix} u_i \\ u_j \\ u_k \end{pmatrix} \tag{1.143}$$

and

$$\mathbf{C} = \begin{pmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{pmatrix} \quad . \tag{1.144}$$

The system of equations (1.142) may be written as

$$\mathbf{u}^e = \mathbf{C}\boldsymbol{\alpha} \tag{1.145}$$

where the only unknown, vector $\boldsymbol{\alpha}$ can be expressed as

$$\boldsymbol{\alpha} = \mathbf{C}^{-1}\mathbf{u}^e \quad . \tag{1.146}$$

Combining Eqs. (1.139) and (1.146) follows that

$$u = \mathbf{M}\mathbf{C}^{-1}\mathbf{u}^e = \mathbf{N}^e\mathbf{u}^e \tag{1.147}$$

where $\mathbf{N}^e$ is the element shape function matrix defined as

$$\mathbf{N}^e = \begin{pmatrix} N_i^e & N_j^e & N_k^e \end{pmatrix} \quad . \tag{1.148}$$

Therefore, Eq. (1.147) can be written as

$$u = N_i^e u_i + N_j^e u_j + N_k^e u_k \tag{1.149}$$

The properties of shape functions are shown in Fig. 1.14. Inverse matrix of C gives

$$\mathbf{C}^{-1} = \frac{1}{2A} \begin{pmatrix} x_j y_k - x_k y_j & x_k y_i - x_i y_k & x_i y_j - x_j y_i \\ y_j - y_k & y_k - y_i & y_i - y_j \\ x_k - x_j & x_i - x_k & x_j - x_i \end{pmatrix} \quad . \tag{1.150}$$

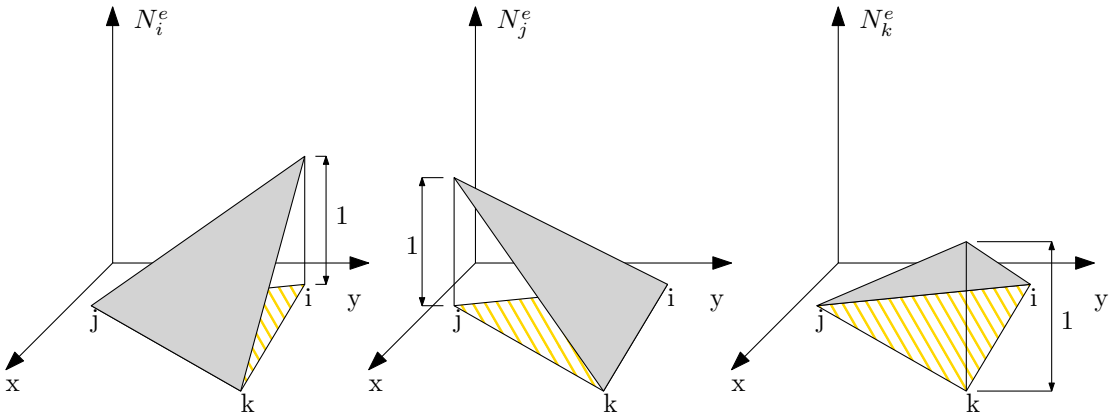


Figure 1.14: Element shape functions

Then, the element shape functions from (1.147) results in

$$\begin{aligned} N_i^e &= \frac{1}{2A}[x_j y_k - x_k y_j + (y_j - y_k)x + (x_k - x_j)y] \\ N_j^e &= \frac{1}{2A}[x_k y_i - x_i y_k + (y_k - y_i)x + (x_i - x_k)y] \\ N_k^e &= \frac{1}{2A}[x_i y_j - x_j y_i + (y_i - y_j)x + (x_j - x_i)y] \end{aligned} \quad . \tag{1.151}$$

The derivative of element shape function matrix is defined according to (1.120) as

$$\mathbf{G}_F^e = \begin{pmatrix} \frac{\partial N_i^e}{\partial x} & \frac{\partial N_j^e}{\partial x} & \frac{\partial N_k^e}{\partial x} \\ \frac{\partial N_i^e}{\partial y} & \frac{\partial N_j^e}{\partial y} & \frac{\partial N_k^e}{\partial y} \end{pmatrix} \quad . \tag{1.152}$$

The element shape functions are given by (1.151). It means that matrix $\mathbf{G}_F^e$ is equal to

$$\mathbf{G}_F^e = \frac{1}{2A} \begin{pmatrix} y_j - y_k & y_k - y_i & y_i - y_j \\ x_k - x_j & x_i - x_k & x_j - x_i \end{pmatrix} \quad . \tag{1.153}$$

The matrix $\mathbf{G}_F^e$ is constant. From (1.135) follows that the stress is constant over the element.

The application TorPy supports the sparse matrix format which enables to solve significantly larger-scale problems compared to the case if dense matrix storage is used. Instead of full global matrix, it is used only three vectors. The example of notation is shown in Fig. 1.16. The sparse matrix for square cross-section with 121 degrees of freedom is shown in Figure 1.15. The non-zero values are marked as blue dots and the dimension of the matrix is 121×121.

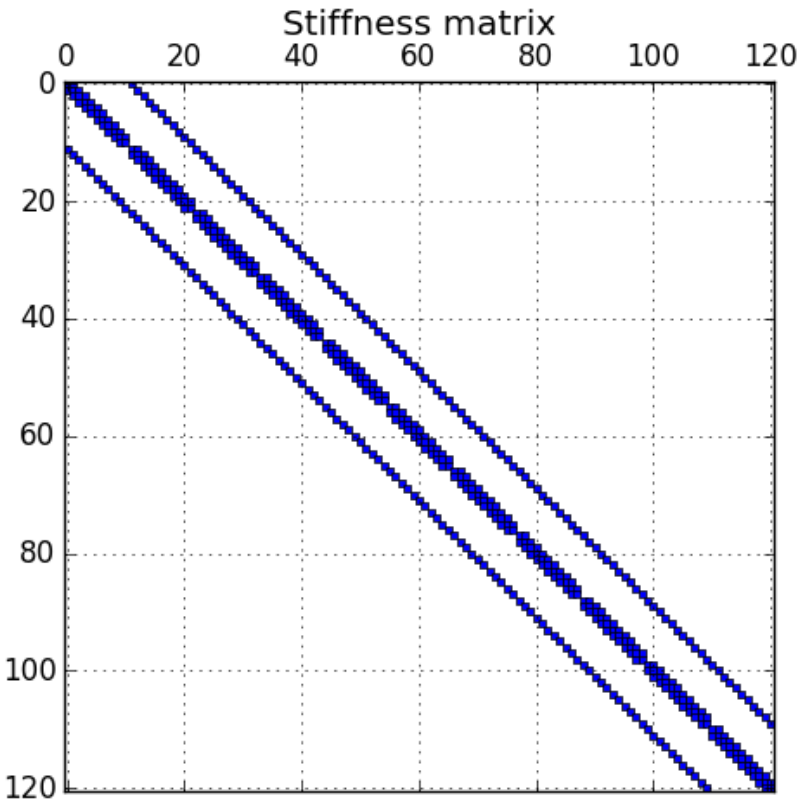


Figure 1.15: Example of a sparse matrix

The reader may compare a dense matrix for five DOF shown in fig. 2.5.

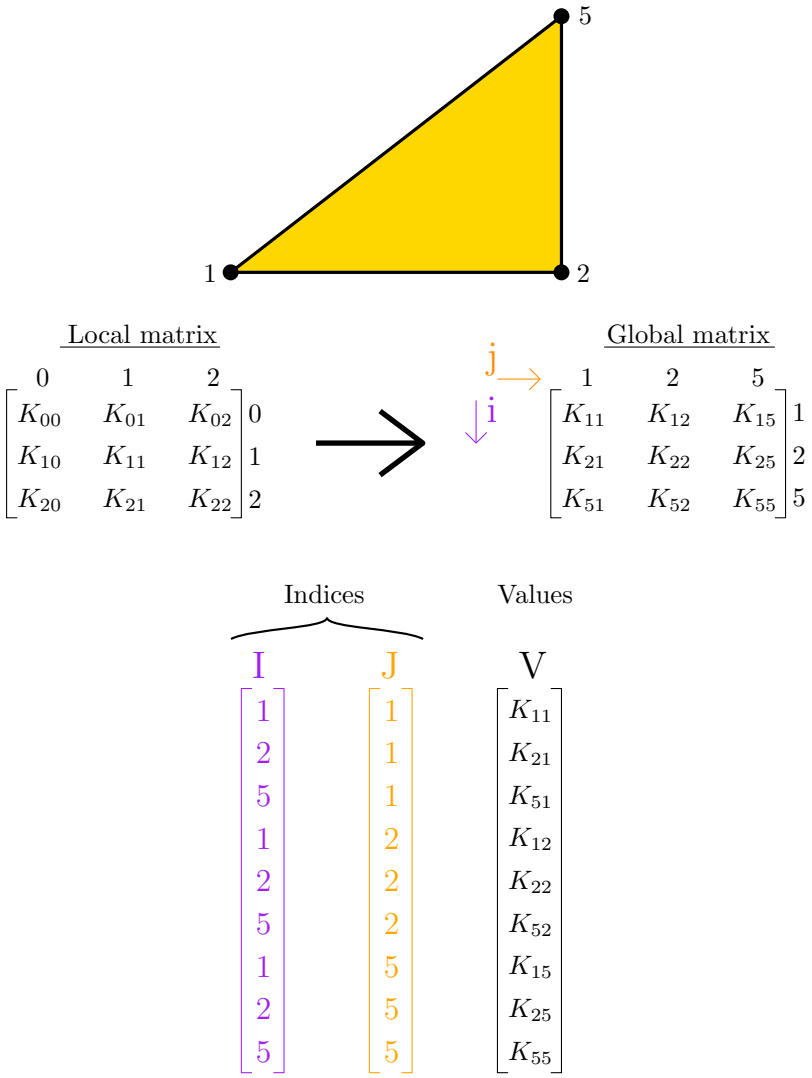


Figure 1.16: Sparse matrix notation

1.4.2 Quadrilateral element

Two types of quadrilateral elements will be introduced. Only isoparametric elements are implemented, however, it is useful to introduce four node bilinear element before.

1.4.2a Four node bilinear element = Lagrange's element

The four node bilinear element is more advanced plane element than tringular, on the other hand, the sides of rectangle must be parallel (see Fig. 1.17). It means that for special shapes of the cross-section, they are not ideal. Similarly to triangular element, the $\mathbf{M}$ matrix is defined as

$$\mathbf{M} = \begin{pmatrix} 1 & x & y & xy \end{pmatrix} \tag{1.154}$$

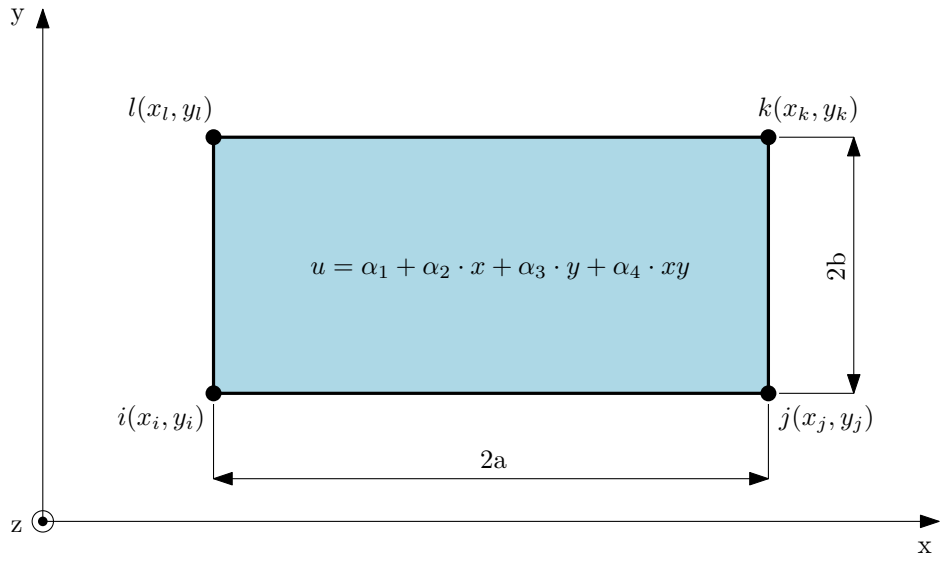


Figure 1.17: Rectangular element

and the matrix of coefficients $\boldsymbol{\alpha}$ is given by

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} . \tag{1.155}$$

It follows that the approximation function can be written as

$$u = \mathbf{N}^e \mathbf{u}^e \tag{1.156}$$

where $\mathbf{N}^e$

$$\begin{aligned} N_i^e &= \frac{1}{4ab}(x - x_j)(y - y_l);\ N_j^e = -\frac{1}{4ab}(x - x_i)(y - y_k) \\ N_k^e &= \frac{1}{4ab}(x - x_l)(y - y_j);\ N_l^e = -\frac{1}{4ab}(x - x_k)(y - y_i) \end{aligned} . \tag{1.157}$$

The matrix $\mathbf{N}^e$ is derived in the same way as in Section 1.4.1.

$$\mathbf{G}_F^e = \begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial x} \\ \dfrac{\partial \mathbf{N}^e}{\partial x} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial N_i^e}{\partial x} & \dfrac{\partial N_j^e}{\partial x} & \dfrac{\partial N_k^e}{\partial x} & \dfrac{\partial N_l^e}{\partial x} \\ \dfrac{\partial N_i^e}{\partial y} & \dfrac{\partial N_j^e}{\partial y} & \dfrac{\partial N_k^e}{\partial y} & \dfrac{\partial N_l^e}{\partial y} \end{pmatrix} . \tag{1.158}$$

According to Equation (1.120), the derivative of the element shape function matrix gives

$$\mathbf{G}_F^e = \begin{pmatrix} y - y_l & y_k - y & y - y_j & y_i - y \\ x - x_j & x_i - x & x - x_l & x_k - x \end{pmatrix} . \tag{1.159}$$

It is clear, that the stress varies linearly over the element. Due to restrictions, it is more convenient to apply isoparametric finite elements.

1.4.2b Quadrilateral isoparametric element

Assume a quadrilateral element as shown in Fig. 1.18.

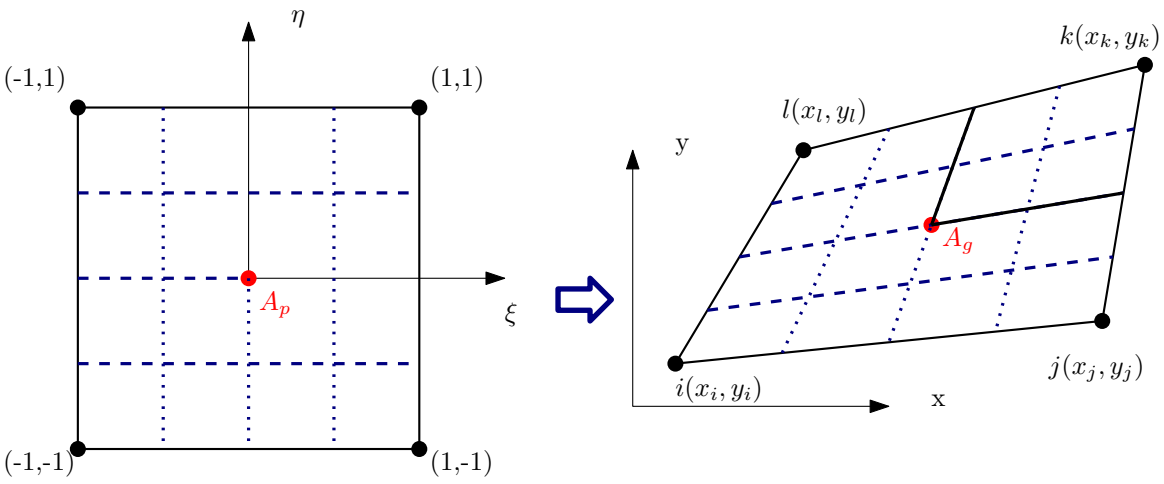


Figure 1.18: Mapping (transformation) into quadrilateral isoparametric element

The aim is to transform (map) the parent domain into arbitrary quadrilateral in global xy domain. However, the mapping must comply with limitations:

- One point from parent domain must correspond to only one point in global domain (e.g. A_p and A_g)
- The outer node from parent domain corresponds to outer node from global domain (i.e. the black dots)

Recall Eq. (1.147), where the unknown field (displacement) u is interpolated between displacement of the nodes $\mathbf{u}^e$. Similarly, the geometry can be transformed as

$$\begin{aligned} x = x(\xi,\eta) &= \mathbf{N}^e(\xi,\eta)\mathbf{x}^e \\ y = y(\xi,\eta) &= \mathbf{N}^e(\xi,\eta)\mathbf{y}^e \end{aligned} \tag{1.160}$$

where coordinates of the nodes in global domain are defined as

$$\mathbf{x}^e = \begin{pmatrix} x_i \\ x_j \\ x_k \\ x_l \end{pmatrix} ; \mathbf{y}^e = \begin{pmatrix} y_i \\ y_j \\ y_k \\ y_l \end{pmatrix} \tag{1.161}$$

and the element shape function matrix is given by

$$\mathbf{N}^e = \begin{pmatrix} N_i^e(\xi,\eta) & N_j^e(\xi,\eta) & N_k^e(\xi,\eta) & N_l^e(\xi,\eta) \end{pmatrix} . \tag{1.162}$$

Referring to Eq. (1.157) and Fig. 1.17, the element shape functions are defined as

$$\begin{aligned} N_i^e(\xi,\eta) &= \frac{1}{4}(\xi-1)(\eta-1); \;\; N_j^e(\xi,\eta) = -\frac{1}{4}(\xi+1)(\eta-1) \\ N_k^e(\xi,\eta) &= \frac{1}{4}(\xi+1)(\eta+1); \;\; N_l^e(\xi,\eta) = -\frac{1}{4}(\xi-1)(\eta+1) \end{aligned} \quad . \tag{1.163}$$

If the relation (1.160) is differentiated, from the chain rule follows that

$$\begin{pmatrix} \mathrm{d}x \\ \mathrm{d}y \end{pmatrix} = \begin{pmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{pmatrix} \begin{pmatrix} \mathrm{d}\xi \\ \mathrm{d}\eta \end{pmatrix} \quad . \tag{1.164}$$

Establish the Jacobian matrix $\mathbf{J}$

$$\mathbf{J} = \begin{pmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{pmatrix} \tag{1.165}$$

where its determinant is so-called Jacobian and it is required that

$$|\mathbf{J}| > 0 \quad . \tag{1.166}$$

If the Eq. (1.166) is fulfilled, the restrictions are satisfied and mapping is one-to-one, i.e. unique. In accordance with Eq. (1.147) implies that

$$u = u(\xi,\eta) = \mathbf{N}^e(\xi,\eta)\mathbf{u}^e \quad . \tag{1.167}$$

The derivative of the element shape function matrix is defined according to (1.120) as

$$\mathbf{G}_F^e = \begin{pmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{pmatrix} \mathbf{N}^e \quad . \tag{1.168}$$

It is convenient to differentiate the element shape function matrix $\mathbf{N}^e$ with respect to ξ and η. This gives

$$\begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial \xi} \\ \dfrac{\partial \mathbf{N}^e}{\partial \eta} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial x}\dfrac{\partial x}{\partial \xi} & + & \dfrac{\partial \mathbf{N}^e}{\partial y}\dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial \mathbf{N}^e}{\partial x}\dfrac{\partial x}{\partial \eta} & + & \dfrac{\partial \mathbf{N}^e}{\partial y}\dfrac{\partial y}{\partial \eta} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \\ \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{pmatrix} \begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial x} \\ \dfrac{\partial \mathbf{N}^e}{\partial y} \end{pmatrix} \quad . \tag{1.169}$$

The Eq. (1.169) is determined for the quadrilateral isoparametric element from Fig. 1.18 as

$$\begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial \xi} \\ \dfrac{\partial \mathbf{N}^e}{\partial \eta} \end{pmatrix} = \frac{1}{4} \begin{pmatrix} \eta-1 & -\eta+1 & \eta+1 & -\eta-1 \\ \xi-1 & -\xi-1 & \xi+1 & -\xi+1 \end{pmatrix} \quad . \tag{1.170}$$

The derivative follows from (1.163). Employing (1.165) into (1.169) gives

$$\begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial \xi} \\ \dfrac{\partial \mathbf{N}^e}{\partial \eta} \end{pmatrix} = \mathbf{J}^\top \begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial x} \\ \dfrac{\partial \mathbf{N}^e}{\partial y} \end{pmatrix} \quad . \tag{1.171}$$

With (1.171) and (1.168) it is obtained

$$\mathbf{G}_F^e = \left(\mathbf{J}^\top\right)^{-1} \begin{pmatrix} \dfrac{\partial}{\partial \xi} \\ \dfrac{\partial}{\partial \eta} \end{pmatrix} \mathbf{N}^e = \left(\mathbf{J}^\top\right)^{-1} \begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial \xi} \\ \dfrac{\partial \mathbf{N}^e}{\partial \eta} \end{pmatrix} \quad . \tag{1.172}$$

Using Eq. (1.172) into (1.128) yields

$$\mathbf{K}^e = \int_{-1}^{1} \int_{-1}^{1} \begin{pmatrix} \dfrac{\partial \mathbf{N}^{e\top}}{\partial \xi} & \dfrac{\partial \mathbf{N}^{e\top}}{\partial \eta} \end{pmatrix} \mathbf{J}^{-1} \mathbf{D} \left(\mathbf{J}^\top\right)^{-1} \begin{pmatrix} \dfrac{\partial \mathbf{N}^e}{\partial \xi} \\ \dfrac{\partial \mathbf{N}^e}{\partial \eta} \end{pmatrix} |\mathbf{J}| d\xi d\eta \tag{1.173}$$

where the expression $|\mathbf{J}|\mathrm{d}\xi\mathrm{d}\eta$ corresponds to dA. The numerical integration is used for evaluation of the stiffness matrix, since the analytical integration for isoparametric elements is complicated.

1.4.3 Solving system of linear equations

In previous section, the Eq. (1.132) was derived

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{1.174}$$

where dimensions of the matrices appertain to $\mathbf{K} \in \mathbb{R}^{n \times n}$, $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{f} \in \mathbb{R}^n$. Important properties of the stiffness matrix $\mathbf{K}$ are:

- positive-definite ($\mathbf{u}^\top \mathbf{K}\mathbf{u} > 0$)
- symmetric ($\mathbf{K}^\top = \mathbf{K}$)
- singular - if the boundary conditions are not applied
- sparse

It was emphasized that sparse matrix format is more efficient and allows to solve significantly larger problems. There exists a lot of solutions and two of them were implemented.

1.4.3a Lagrange multipliers

The original system of linear equations (1.174) is enlarged and the new form is defined by

$$\begin{pmatrix} \mathbf{K} & \mathbf{B}^\top \\ \mathbf{B} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{c} \end{pmatrix} \tag{1.175}$$

where the dimensions belong to $\mathbf{B} \in \mathbb{R}^{m \times n}$, $\boldsymbol{\lambda} \in \mathbb{R}^m$, $\mathbf{0} \in \mathbb{R}^{m \times m}$ and $\mathbf{c} \in \mathbb{R}^m$. The dimension n is the number of degrees of freedom and m is the number of constraints. The matrix $\mathbf{B}$ is so-called boolean matrix, the vector $\boldsymbol{\lambda}$ carries multipliers, the matrix $\mathbf{0}$ represents zeros and $\mathbf{c}$ is vector of constants. The formula (1.175) can be rewritten as

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.176}$$

where $\mathbf{x}$ is vector of unknowns and $\mathbf{A}$ is called stiffness-bordered matrix. This system of linear equations is solved by Gauss elimination method. The equation

$$\mathbf{B}\mathbf{u} - \mathbf{c} = \mathbf{0} \tag{1.177}$$

is so-called constraint equation [10] and vector $\mathbf{c} = \mathbf{0}$ due to (1.84). The method is illustrated in Fig. 1.19. A domain of membrane is divided into four elements with nine nodes. Eight of them is constrained. The number of nodes is equal to degrees of freedom. This gives n = 9 and m = 8.

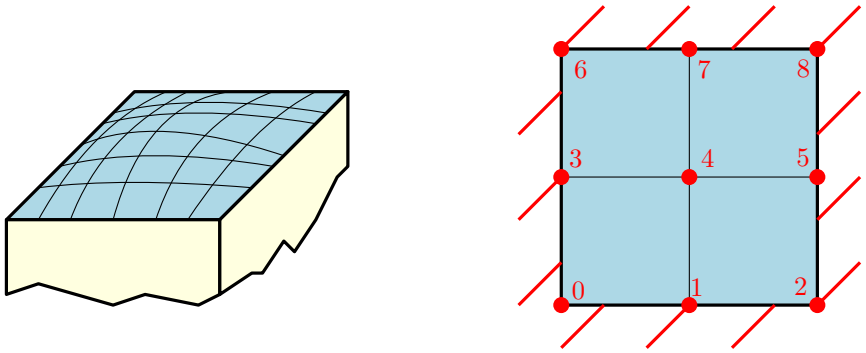


Figure 1.19: Membrane and its discretization

It means that boolean matrix $\mathbf{B} \in \mathbb{R}^{8 \times 9}$ and is given by

$$\mathbf{B} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} . \tag{1.178}$$

1.4.3b Conjugate gradient method

The conjugate gradient method (CGM) can be either a direct or an iterative method. The advantage of this method represents application to sparse systems. CGM is often used for solving of large system of linear equations or optimization problems. The method is implemented to solve system (1.132)

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{1.179}$$

where matrix $\mathbf{K}$ is not singular (the boundary conditions have to be apllied). The system of linear equations can be transformed into function that will be minimized

$$F(\hat{\mathbf{u}}) = \frac{1}{2}\left(\hat{\mathbf{u}}, \mathbf{K}\hat{\mathbf{u}}\right) - \left(\hat{\mathbf{u}}, \mathbf{f}\right) \quad . \tag{1.180}$$

It exists $\hat{\mathbf{u}} = \mathbf{u}$, which is the minimum of function as well as the solution of Eq. (1.180). CGM is implemented as iterative method.

A derivative of function F (1.180) gives the gradient (residual) at i-th step

$$\mathbf{g}_i = \mathbf{K}\hat{\mathbf{u}}_\mathbf{i} - \mathbf{f} \tag{1.181}$$

and it must be chosen an initial guess $\hat{\mathbf{u}}_\mathbf{o}$ for the first step. It can be vector of zeros. The initial gradient is then

$$\mathbf{g}_0 = \mathbf{K}\hat{\mathbf{u}}_\mathbf{0} - \mathbf{f} \neq \mathbf{0} \tag{1.182}$$

and it is difference between internal and external forces. For exact solution, the residual can be equal to vector of zeros, therefore $\mathbf{g}_i = \mathbf{0}$. In general, the initial gradient is non-zero as written in Eq. (1.182). A preconditioning is given by diagonal matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$, $P_{j,j} = 1/K_{j,j}, j = 0, 1, ..., n-1$, i.e.

$$\mathbf{K} = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 6 & 0 \\ 1 & 0 & 3 \end{pmatrix}; \mathbf{P} = \begin{pmatrix} 1/4 & 0 & 0 \\ 0 & 1/6 & 0 \\ 0 & 0 & 1/3 \end{pmatrix} \tag{1.183}$$

where diagonal elements of matrix $\mathbf{K}$ are reciprocal. This type of preconditioner is the simplest way. The optimum location of the solution is defined by

$$\mathbf{u}_i = \mathbf{u}_{i-1} + \alpha_i \mathbf{p}_{i-1} \tag{1.184}$$

where $\mathbf{p}_{i-1}$ is conjugate vector which determines the direction and the coefficient α_i is calculated as

$$\alpha_i = -\frac{\mathbf{g}_{i-1}^\top \cdot \mathbf{z}_{i-1}}{\mathbf{p}_{i-1}^\top \cdot \mathbf{K} \cdot \mathbf{p}_i} \tag{1.185}$$

and it represents a length of step. The parameter β_i is given by

$$\beta_i = \frac{\mathbf{z}_i^\top \cdot \mathbf{g}_i}{\mathbf{z}_{i-1}^\top \cdot \mathbf{g}_{i-1}} \tag{1.186}$$

in accordance with Fletcher - Reeves formula. The algorithm of preconditioned conjugate gradient method is in Figure 1.20. Detailed information are in [17],[27] or [15].

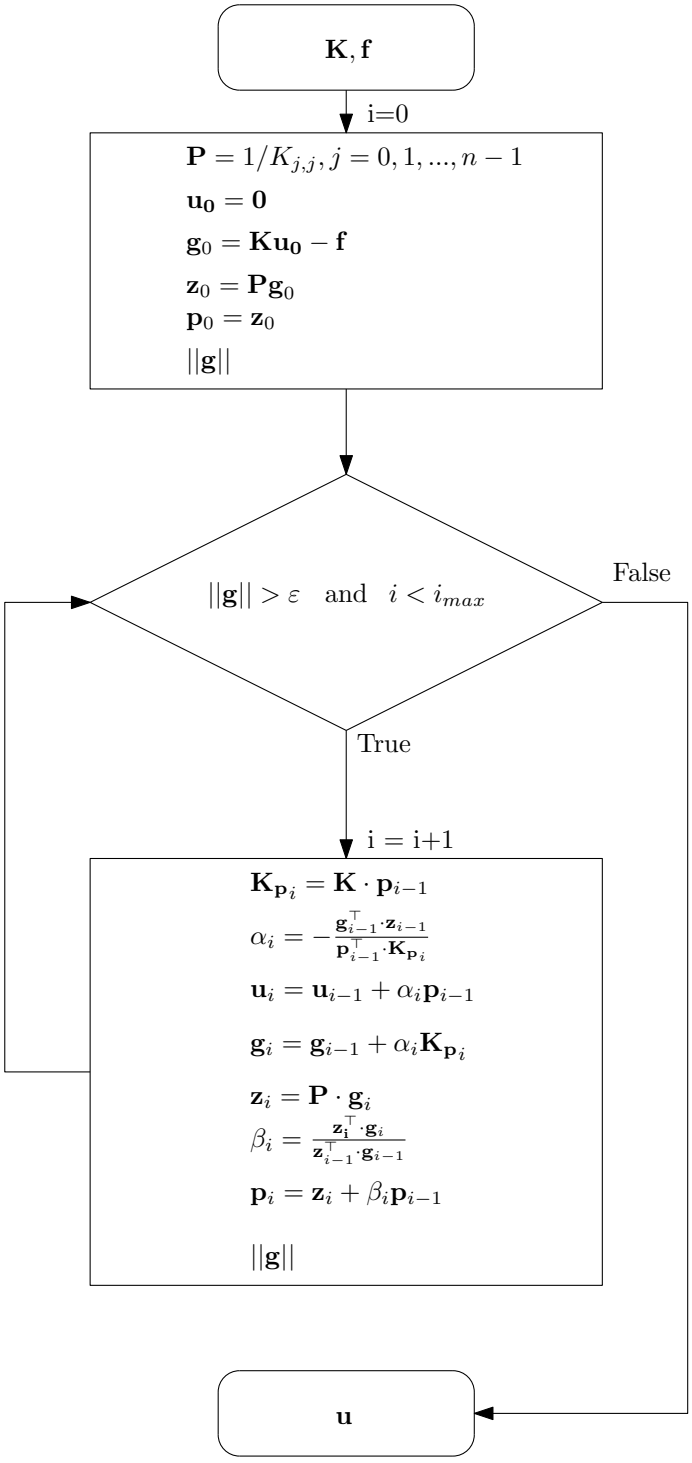


Figure 1.20: CGM algorithm

1.5 The boundary element method

It is considered a modified equation (1.64) with unit right-hand side and the boundary condition (1.84). It can be written as

$$\boxed{\begin{aligned} -\Delta u &= 1 \\ u_\Gamma &= 0 \end{aligned}} \tag{1.187}$$
$$\tag{1.188}$$

where u_Γ determines u along boundary. From (1.64) and (1.84) also results a relation

$$\phi(x,y) = u(x,y)2G\vartheta \tag{1.189}$$

that connect variables u and ϕ. The solution u can be divided into two parts as

$$u = u_0 + \tilde{u} \tag{1.190}$$

that represent particular and homogeneous parts. Suppose the solution of u_0 as

$$u_0 = -\frac{1}{4}\left(x^2 + y^2\right) \tag{1.191}$$

and the second-order partial derivatives give

$$\Delta u_0 = \frac{\partial^2 u_0}{\partial x^2} + \frac{\partial^2 u_0}{\partial y^2} = -1 \ . \tag{1.192}$$

Combining Eqs. (1.190) and (1.187) it follows that

$$-\Delta\left(u_0 + \tilde{u}\right) = 1 \tag{1.193}$$

that is equal to formula

$$-\Delta u_o - \Delta\tilde{u} = 1 \ . \tag{1.194}$$

Insertion of (1.192) into (1.194) yields

$$1 - \Delta\tilde{u} = 1 \tag{1.195}$$

or

$$\Delta\tilde{u} = 0 \tag{1.196}$$

i.e. the Poisson equation is transformed into Laplace equation. It remains to investigate boundary conditions. The solution on the boundary can be split as well into two parts and in accordance with (1.188) it is equal to zero. The equation is written as

$$u_\Gamma = u_{o\Gamma} + \tilde{u_\Gamma} = 0 \tag{1.197}$$

and with (1.191) it is obvious that

$$\tilde{u_\Gamma} = -u_{o\Gamma} = \frac{1}{4}\left(x_\Gamma^2 + y_\Gamma^2\right) \ . \tag{1.198}$$

1.5.1 Laplace equation

Instead of Poisson equation, it is possible to solve Laplace equation simple trick. Summarization

$$\boxed{\Delta\tilde{u} = 0} \tag{1.199}$$

$$\tilde{u_\Gamma} = \frac{1}{4}\left(x_\Gamma^2 + y_\Gamma^2\right) \tag{1.200}$$

With (1.190) and (1.191) implies that

$$u(x,y) = \tilde{u}(x,y) - \frac{1}{4}\left(x^2 + y^2\right) \tag{1.201}$$

where $u(x,y)$ is a continuous function, $\tilde{u}(x,y)$ is the unknown function and x and y are coordinates in the domain A. The domain (see Fig. 1.21) may represent arbitrary cross-section.

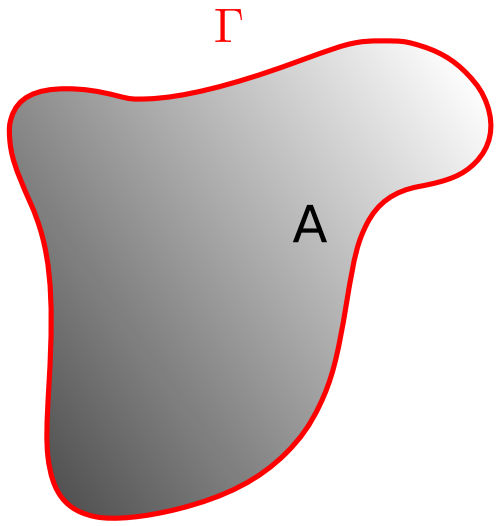


Figure 1.21: Domain A and boundary Γ

1.5.1a Inverse formulation

The boundary element formulation will be carried out for $\tilde{u}$. General notation of (1.199) may look as

$$\Delta\tilde{u} = 0 \quad in\ A \tag{1.202}$$

with boundary conditions

$$\begin{aligned} &\text{a) Essential (Dirichlet) Conditions } \tilde{u} = \bar{u} \text{ on } \Gamma_1 \\ &\text{b) Natural (Neumann) Conditions } q = \frac{\partial\tilde{u}}{\partial n} = \bar{q} \text{ on } \Gamma_2 \end{aligned} \tag{1.203}$$

where $\Gamma = \Gamma_1 + \Gamma_2$. The first step is the same as in finite element method (see (1.107)). The equation (1.202) is multiplied by weight (test) function and integrate over area A, i.e.

$$\iint_A (\Delta\tilde{u})\, v \, \mathrm{d}A = 0 \ . \tag{1.204}$$

From Green - Gauss theorem (integration by parts in several dimensions) yields

$$\iint_A (\Delta\tilde{u})\, v \, \mathrm{d}A = \int_\Gamma \frac{\partial\tilde{u}}{\partial n} v \, \mathrm{d}\Gamma - \iint_A \nabla\tilde{u}\nabla v \, \mathrm{d}A \tag{1.205}$$

and the integration by parts again of the second term from (1.205) leads to

$$\iint_A \nabla\tilde{u}\nabla v \, \mathrm{d}A = \int_\Gamma \frac{\partial v}{\partial n}\tilde{u} \, \mathrm{d}\Gamma - \iint_A \tilde{u}\Delta v \, \mathrm{d}A \ . \tag{1.206}$$

Insertion of (1.206) in (1.205) gives

$$\iint_A (\Delta\tilde{u})\, v \, \mathrm{d}A = \int_\Gamma \frac{\partial\tilde{u}}{\partial n} v \, \mathrm{d}\Gamma - \int_\Gamma \tilde{u}\frac{\partial v}{\partial n} \, \mathrm{d}\Gamma + \iint_A \tilde{u}\Delta v \, \mathrm{d}A \ . \tag{1.207}$$

Equation (1.207) can be rewritten as

$$\iint_A (\Delta\tilde{u})\, v \, \mathrm{d}A - \iint_A \tilde{u}\,(\Delta v) \ \mathrm{d}A = \int_\Gamma \frac{\partial\tilde{u}}{\partial n} v \, \mathrm{d}\Gamma - \int_\Gamma \tilde{u}\frac{\partial v}{\partial n} \, \mathrm{d}\Gamma \tag{1.208}$$

or

$$\iint_A (\Delta\tilde{u})\, v \, \mathrm{d}A - \iint_A \tilde{u}\,(\Delta v) \ \mathrm{d}A = \int_\Gamma \left(\frac{\partial\tilde{u}}{\partial n} v - \tilde{u}\frac{\partial v}{\partial n}\right) \mathrm{d}\Gamma \tag{1.209}$$

and the equation is so-called Green's second identity. The boundary conditions remains

$$\begin{aligned} &\text{a)}\ \tilde{u} = \bar{u} && \text{on } \Gamma_1 \\ &\text{b)}\ q = \frac{\partial\tilde{u}}{\partial n} = \bar{q} && \text{on } \Gamma_2 \end{aligned} \ . \tag{1.210}$$

Using boundary conditions (1.210) and (1.202) follows that

$$\iint_A \tilde{u}\,(\Delta v) \ \mathrm{d}A = -\int_{\Gamma_1} qv \, \mathrm{d}\Gamma_1 - \int_{\Gamma_2} \bar{q}v \, \mathrm{d}\Gamma_2 + \int_{\Gamma_1} \bar{u}\frac{\partial v}{\partial n} \, \mathrm{d}\Gamma_1 + \int_{\Gamma_2} \tilde{u}\frac{\partial v}{\partial n} \, \mathrm{d}\Gamma_2 \ . \tag{1.211}$$

The aim is to eliminate the domain term (left-hand side of Eq. (1.211)) and convert the entire domain only to the boundary Γ. For this purpose it is employed Dirac's delta function (see Appendix A) as

$$\Delta v = -\delta(x, \xi) \tag{1.212}$$

where v is fundamental solution, often termed as u^*. Multiplying the Dirac distribution (1.212) by other function implies that

$$\iint_A \tilde{u}\,(\Delta v) \ \mathrm{d}A = -\iint_A \tilde{u}\delta(x, \xi) \, \mathrm{d}A = -\tilde{u}(\xi) \tag{1.213}$$

where x denotes field point vector and ξ is load point vector. The sifting property of Dirac distribution are

$$\iint_A \tilde{u}(x)\delta(x,\xi)dA = \begin{cases} \tilde{u}(\xi) & \text{for } \xi \in A \\ 0 & \text{for } \xi \notin A, \xi \notin \Gamma \\ \text{undef} & \text{for } \xi \in \Gamma \end{cases} \tag{1.214}$$

which means that function $\tilde{u}(\xi)$ is defined only when the load point ξ lies inside the domain A. Using fundamental solution as the weight function and employing (1.213) follows that

$$-\tilde{u}(\xi) = -\int_{\Gamma_1} qu^* \,\mathrm{d}\Gamma_1 - \int_{\Gamma_2} \bar{q}u^* \,\mathrm{d}\Gamma_2 + \int_{\Gamma_1} \bar{u}q^* \,\mathrm{d}\Gamma_1 + \int_{\Gamma_2} \tilde{u}q^* \,\mathrm{d}\Gamma_2 \tag{1.215}$$

or

$$\tilde{u}(\xi) = \int_{\Gamma} qu^* \,\mathrm{d}\Gamma - \int_{\Gamma} \tilde{u}q^* \,\mathrm{d}\Gamma \tag{1.216}$$

where q^* is termed the derivative of fundamental solution. Note that in FEM, the test (weight) function is chosen as piecewise polynomials, whereas in BEM, the fundamental solution is chosen. Equation (1.216) is called representation formula. If the potential $\tilde{u}$ and its derivative q are known on the boundary, the equation returns unknown potential $\tilde{u}$ inside domain ($\xi \in A$).

1.5.1b Fundamental solutions

The fundamental solution of Eq. (1.212) for isotropic two dimensional medium is defined by

$$u^*(x,\xi) = -\frac{1}{2\pi} \ln |\mathbf{r}| \tag{1.217}$$

$$q^*(x,\xi) = -\frac{\mathbf{r}\cdot\mathbf{n}}{2\pi|\mathbf{r}|^2} \tag{1.218}$$

where $\mathbf{r} = |\mathbf{x} - \boldsymbol{\xi}|$ is called Euclidean distance.

1.5.1c Boundary integral equation

As mentioned before, the objective is to acquire the equation that contains only boundary inputs. According to (1.214), it is essential to move the load point ξ to the boundary. The limits and the modification of the boundary in the vicinity of the load point ξ will be used. Assume the load point ξ lying on boundary and small disk with radius ε with center in point ξ, see Figure 1.22. The modified boundary Γ' consists of original boundary Γ and Γ_ε minus Γ_ε^*, mathematically written as

$$\Gamma = \Gamma' + \Gamma_\varepsilon^* - \Gamma_\varepsilon \tag{1.219}$$

and the limit is is equal to

$$\Gamma = \lim_{\varepsilon \to 0} \Gamma' \ . \tag{1.220}$$

Due to boundary extension, the load point ξ lies inside domain and the representation formula is valid. The element dF_ε can be expressed according to Figure 1.22 as

$$\mathrm{d}\Gamma_\varepsilon = \varepsilon \,\mathrm{d}\theta \tag{1.221}$$

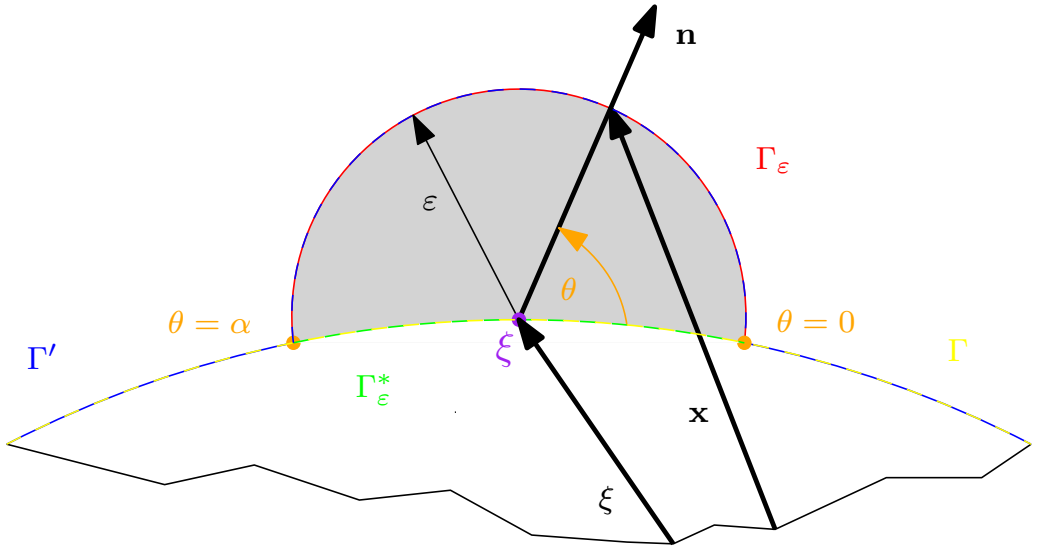


Figure 1.22: Investigation of a load point ξ on boundary Γ

where

$$\varepsilon = |\mathbf{x} - \boldsymbol{\xi}| \tag{1.222}$$

is also the Euclidean distance between the load point $\boldsymbol{\xi}$ and the field point $\mathbf{x}$. This process is important in the limiting process when the load point is moved to the boundary. Consequently, it is possible to investigate integrals in equation (1.216). The first term on right-hand side and the fundamental solution (1.217) yields

$$\begin{aligned} \int_\Gamma q u^* \, \mathrm{d}\Gamma &= -\lim_{\varepsilon \to 0} \int_{\Gamma'} q \frac{\ln |\mathbf{x} - \boldsymbol{\xi}|}{2\pi} \, \mathrm{d}\Gamma = \\ &= -\lim_{\varepsilon \to 0} \int_{\Gamma - \Gamma_\varepsilon^*} q \frac{\ln |\mathbf{x} - \boldsymbol{\xi}|}{2\pi} \, \mathrm{d}\Gamma - \lim_{\varepsilon \to 0} \int_{\Gamma_\varepsilon} q \frac{\ln |\mathbf{x} - \boldsymbol{\xi}|}{2\pi} \, \mathrm{d}\Gamma \end{aligned} \tag{1.223}$$

where the first integral is weakly singular. It means that

$$\lim_{\varepsilon \to 0} \int_{\Gamma - \Gamma_\varepsilon^*} q \frac{\ln |\mathbf{x} - \boldsymbol{\xi}|}{2\pi} \, \mathrm{d}\Gamma = \int_\Gamma q \frac{\ln |\mathbf{x} - \boldsymbol{\xi}|}{2\pi} \, \mathrm{d}\Gamma \ . \tag{1.224}$$

The other integral with combination of (1.221) and (1.222) is given by

$$\lim_{\varepsilon \to 0} \int_{\Gamma_\varepsilon} q \frac{\ln |\mathbf{x} - \boldsymbol{\xi}|}{2\pi} \, \mathrm{d}\Gamma = \lim_{\varepsilon \to 0} \int_{\theta=0}^{\alpha} q \frac{\ln(\varepsilon)}{2\pi} \varepsilon \, \mathrm{d}\theta \tag{1.225}$$

and employing l'Hospital's rule it is obtained

$$\lim_{\varepsilon \to 0} \frac{1}{2\pi} \int_{\theta=0}^{\alpha} q \frac{\ln(\varepsilon)'}{\left(\frac{1}{\varepsilon}\right)'} \varepsilon \, \mathrm{d}\theta = \lim_{\varepsilon \to 0} \frac{1}{2\pi} \int_{\theta=0}^{\alpha} -q\varepsilon \, \mathrm{d}\theta = 0 \ . \tag{1.226}$$

The second term from (1.216) is so-called strongly singular integral and with (1.218) implies that

$$\begin{aligned} -\int_{\Gamma} \tilde{u} q^* \,\mathrm{d}\Gamma &= \lim_{\varepsilon\to 0} \int_{\Gamma'} \tilde{u} \frac{(\mathbf{x}-\boldsymbol{\xi})\mathbf{n}}{2\pi|\mathbf{x}-\boldsymbol{\xi}|^2} \,\mathrm{d}\Gamma = \\ &= \lim_{\varepsilon\to 0} \int_{\Gamma-\Gamma_\varepsilon^*} \tilde{u} \frac{(\mathbf{x}-\boldsymbol{\xi})\mathbf{n}}{2\pi|\mathbf{x}-\boldsymbol{\xi}|^2} \,\mathrm{d}\Gamma + \lim_{\varepsilon\to 0} \int_{\Gamma_\varepsilon} \tilde{u} \frac{(\mathbf{x}-\boldsymbol{\xi})n_i}{2\pi|\mathbf{x}-\boldsymbol{\xi}|^2} \,\mathrm{d}\Gamma \ . \end{aligned} \tag{1.227}$$

The first integral from (1.227) corresponds to Cauchy principal value, i.e.

$$\lim_{\varepsilon\to 0} \int_{\Gamma-\Gamma_\varepsilon^*} \tilde{u} q^* \,\mathrm{d}\Gamma = \cancel{\int_{\Gamma}} \tilde{u} q^* \,\mathrm{d}\Gamma \ . \tag{1.228}$$

The other integral from (1.227) may be written as

$$\lim_{\varepsilon\to 0} \int_{\Gamma_\varepsilon} \tilde{u} \frac{(\mathbf{x}-\boldsymbol{\xi})\mathbf{n}}{2\pi|\mathbf{x}-\boldsymbol{\xi}|^2} \,\mathrm{d}\Gamma = \lim_{\varepsilon\to 0} \int_{\theta=0}^{\alpha} \tilde{u} \frac{\varepsilon}{2\pi\varepsilon^2} \varepsilon \,\mathrm{d}\theta = \tilde{u}(\xi) \int_{\theta=0}^{\alpha} \frac{1}{2\pi} \,\mathrm{d}\theta = \frac{\alpha}{2\pi} \tilde{u}(\xi) \tag{1.229}$$

and clearly, the expression (1.229) is not equal to zero as (1.226). Equation (1.216) with (1.224), (1.228) and (1.229) becomes

$$\tilde{u}(\xi) = \frac{\alpha}{2\pi} \tilde{u}(\xi) + \cancel{\int_{\Gamma}} \tilde{u}(x) \frac{(\mathbf{x}-\boldsymbol{\xi})\mathbf{n}}{2\pi|\mathbf{x}-\boldsymbol{\xi}|^2} \,\mathrm{d}\Gamma - \int_{\Gamma} q(x) \frac{\ln|\mathbf{x}-\boldsymbol{\xi}|}{2\pi} \,\mathrm{d}\Gamma \tag{1.230}$$

or using (1.217) and (1.218) follows that

$$\left(1 - \frac{\alpha}{2\pi}\right) \tilde{u}(\xi) = -\cancel{\int_{\Gamma}} \tilde{u} q^* \,\mathrm{d}\Gamma + \int_{\Gamma} q u^* \,\mathrm{d}\Gamma \tag{1.231}$$

where

$$\left(1 - \frac{\alpha}{2\pi}\right) = c(\xi) \tag{1.232}$$

is called free term coefficient and its properties are defined by

$$c(\xi) = \begin{cases} 1 - \frac{\alpha}{2\pi} & \text{for } \xi \in \Gamma, \Gamma \text{ is not smooth} \\ 1 & \text{for } \xi \in A \\ \frac{1}{2} & \text{for } \xi \in \Gamma, \Gamma \text{ is smooth} \\ 0 & \text{for } \xi \notin \Gamma, \xi \notin A \end{cases} \ . \tag{1.233}$$

Equation (1.231) is so-called boundary integral equation and its solution is known only for simple cases. For more general cases it is essential to split the boundary into elements. This process is very similar to finite element discretisation.

1.5.1d Discretisation of the boundary

The continuous boundary Γ is divided into discrete boundary elements $\Gamma_1, \Gamma_2, ..., \Gamma_n$, where n denotes total number of elements. Consider two-dimensional domain in Fig. 1.23 that is illustrated by fat line. The discretisation is carried out constant shape function and linear shape function. The constant elements determined by blue crosses with red nodes. The red

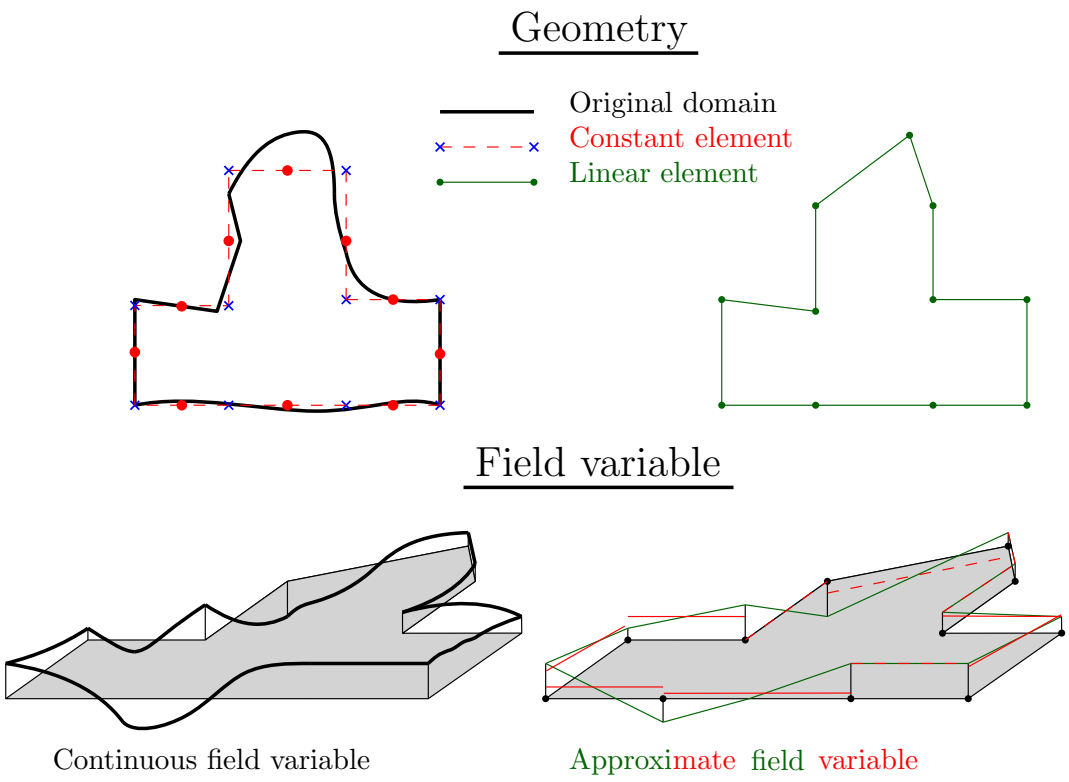


Figure 1.23: Approximations of geometry and unknown field

dashed line replaces the original domain. The same process is used with linear elements. One linear element uses two nodes (green discs). Similarly, a field variable is interpolated. The field variable may be potential (displacement, temperature,...) or its derivation (stress, flux,...). Assume a local coordinate s and a scalar function $\tilde{u}$ that may represent displacement of the membrane. Establish a discrete boundary consists of n elements and interpolate the continuous scalar function as

$$\tilde{u}^e(s) = \sum_{i=1}^{n_e} N_i^e \hat{u}_i^e;\ q^e(s) = \sum_{i=1}^{n_e} N_i^e \hat{q}_i^e \tag{1.234}$$

where n_e corresponds the number of unknowns for one element. N_i^e is called shape function and $\hat{u}_i^e$ is a value of the displacement in node. A matrix notation is written as

$$\tilde{u}^e(s) = \mathbf{N}^e \hat{\mathbf{u}}^e \tag{1.235}$$

where

$$\mathbf{N}^e = \begin{pmatrix} N_i^e & N_j^e & \cdots & N_{n_e}^e \end{pmatrix};\ \hat{\mathbf{u}}^e = \begin{pmatrix} \hat{u}_i \\ \hat{u}_j \\ \vdots \\ \hat{u}_{n_e} \end{pmatrix} . \tag{1.236}$$

The simplest possible approximation in two-dimensional analysis is constant shape function. For this type of function is n_e equal to one and the value of the field function is a constant

over element. The constant shape function has one node in the middle of length of element as shown in Figure 1.24. According to (1.234) it follows that

$$\tilde{u}^e(s) = N_1^e \hat{u}_1^e = \hat{u}_1^e \ . \tag{1.237}$$

The linear shape function approximates the original function more accurate than the constant shape function. It possess two nodes per element (see 1.24). The equation for interpolation is given by

$$\tilde{u}^e(s) = N_1^e \hat{u}_1^e + N_2^e \hat{u}_2^e \tag{1.238}$$

or in matrix notation

$$\tilde{u}^e(s) = \begin{pmatrix} N_1^e & N_2^e \end{pmatrix} \begin{pmatrix} \hat{u}_1 \\ \hat{u}_1 \end{pmatrix} = \mathbf{N}^e \hat{\mathbf{u}}^e \tag{1.239}$$

where

$$N_1^e = 1 - \frac{s}{L^e};\ N_2^e = \frac{s}{L^e} \ . \tag{1.240}$$

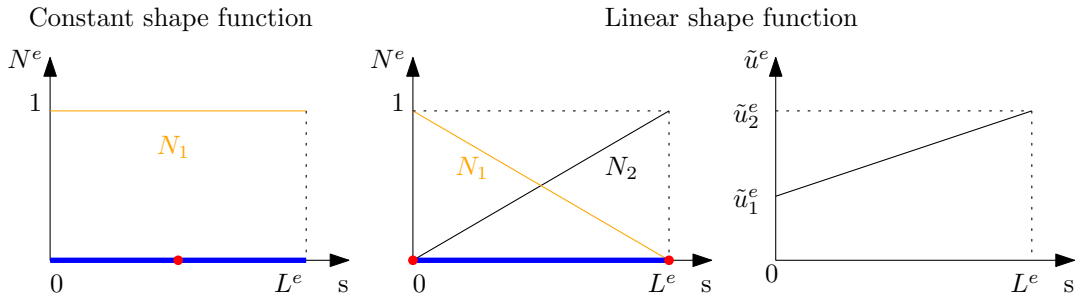


Figure 1.24: One-dimensional shape functions

Use of (1.234) and (1.232) in (1.231) results in

$$c(\xi)\tilde{u}(\xi) = -\sum_{e=1}^{n} \left(\sum_{i=1}^{n_e} \hat{u}_i^e \int_{\Gamma^e} N_i^e q^* \, \mathrm{d}\Gamma \right) + \sum_{e=1}^{n} \left(\sum_{i=1}^{n_e} \hat{q}_i^e \int_{\Gamma^e} N_i^e u^* \, \mathrm{d}\Gamma \right) \tag{1.241}$$

since the field variables $\hat{u}_i^e$ and $\hat{q}_i^e$ in nodes are constants. In this thesis, only constant elements will be implemented and the nodes will be located in the middle of elements. Thus, the angle α is equal to π and the free term coefficient gives

$$c(\xi) = \left(1 - \frac{\pi}{2\pi}\right) = \frac{1}{2} \tag{1.242}$$

and vector $\mathbf{r}$ forms with normal $\mathbf{n}$ right angle, therefore the scalar product gives

$$(\mathbf{x} - \boldsymbol{\xi})\mathbf{n} = 0 \ . \tag{1.243}$$

The boundary integral equation is now discretised and the computation of unknown boundary values may be performed by several ways employing method of weighted residuals. The most often method for the determination of unknowns is the collocation method.

1.5.1e The Collocation Method

The load point ξ is placed one by one on all nodes. For example, the first node gives

$$\begin{matrix} \hat{u}_1 \left(c_1 + \int_{\Gamma^{e,1}} N_1 q^*(x,\xi_1)\,\mathrm{d}\Gamma \right) + \cdots + \hat{u}_n \left(\int_{\Gamma^{e,n}} N_n q^*(x,\xi_1)\,\mathrm{d}\Gamma \right) = \\ \hat{q}_1 \left(\int_{\Gamma^{e,1}} N_1 u^*(x,\xi_1)\,\mathrm{d}\Gamma \right) + \cdots + \hat{q}_n \left(\int_{\Gamma^{e,n}} N_n u^*(x,\xi_1)\,\mathrm{d}\Gamma \right) \end{matrix} . \tag{1.244}$$

In the same manner, every node is collocated and this gives the system of linear equations. It is convenient to establish a substitution as

$$\begin{aligned} \hat{H}_{ij} &= \int_{\Gamma^{e,j}} N_j q^*(x,\xi_i)\,\mathrm{d}\Gamma \\ G_{ij} &= \int_{\Gamma^{e,j}} N_j u^*(x,\xi_i)\,\mathrm{d}\Gamma \end{aligned} \tag{1.245}$$

and this gives

$$\sum_{j=1}^{n} \hat{u}_j H_{ij} = \sum_{j=1}^{n} hat q_j G_{ij} \tag{1.246}$$

where

$$H_{ij} = \begin{cases} \hat{H}_{ij} & \text{if } i \neq j \\ \hat{H}_{ij} + \frac{1}{2} & \text{if } i = j \end{cases} . \tag{1.247}$$

The matrix notation of the system of equations is

$$\begin{pmatrix} H_{11} & \cdots & H_{1n} \\ \vdots & \ddots & \vdots \\ H_{n1} & \cdots & H_{nn} \end{pmatrix} \begin{pmatrix} \hat{u}_1 \\ \vdots \\ \hat{u}_n \end{pmatrix} = \begin{pmatrix} G_{11} & \cdots & G_{1n} \\ \vdots & \ddots & \vdots \\ G_{n1} & \cdots & G_{nn} \end{pmatrix} \begin{pmatrix} \hat{q}_1 \\ \vdots \\ \hat{q}_n \end{pmatrix} \tag{1.248}$$

or in short

$$\mathbf{H}\hat{\mathbf{u}} = \mathbf{G}\hat{\mathbf{q}} \ . \tag{1.249}$$

If the boundary conditions are applied, the system of equations is reformulated so that unknowns are taken to the left-hand side and this gives the known formula

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.250}$$

where $\mathbf{x}$ is vector of unknowns. The alternative approach than the collocation method can be, for instance, the Galerkin method.

1.5.1f Displacement in domain

When the vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{q}}$ are known, the displacement (potential) $\tilde{u}$ at arbitrary point $\xi_A \in A$ can be calculated from the representation formula (1.216). For constant elements, it is obtained

$$\tilde{u}(\xi_A) = \sum_{e=1}^{n} \hat{q}^e G_{Ae} - \sum_{e=1}^{n} \hat{u}_e H_{Ae} \tag{1.251}$$

where

$$H_{Ae} = \int_{\Gamma_e} \frac{(\mathbf{x}-\boldsymbol{\xi})\mathbf{n}}{2\pi|\mathbf{x}-\boldsymbol{\xi}|^2} d\Gamma \text{ and } G_{Ae} = \int_{\Gamma_e} \frac{\ln|\mathbf{x}-\boldsymbol{\xi}|}{2\pi} d\Gamma \ . \tag{1.252}$$

It is suitable to establish matrices

$$\mathbf{H}_A^\top = \begin{pmatrix} H_{11} & H_{12} & \cdots & H_{1n} \end{pmatrix}; \ \mathbf{G}_A^\top = \begin{pmatrix} G_{11} & G_{12} & \cdots & G_{1n} \end{pmatrix} \tag{1.253}$$

so that

$$\tilde{u}(\xi_A) = \mathbf{G}_A^\top \hat{\mathbf{q}} - \mathbf{H}_A^\top \hat{\mathbf{u}} \tag{1.254}$$

thus, the displacement of membrane is calculated as

$$\phi = \left[\tilde{u}(\xi_A) - \frac{1}{4}\left(x_A^2 + y_A^2\right) \right] 2 \cdot G \cdot \vartheta \tag{1.255}$$

where x_A and y_A are coordinates of the point inside domain. The reader can find detailed description of the boundary element method in [13], [20],[4] or [8]. The theory is applied on example in Chapter 2 as well as the finite element theory.

1.5.1g Calculation of element normals

Due to the theory, it is necessary to know a normal of every element. The calculation is illustrated on example. Assume a domain in Fig. 1.25.

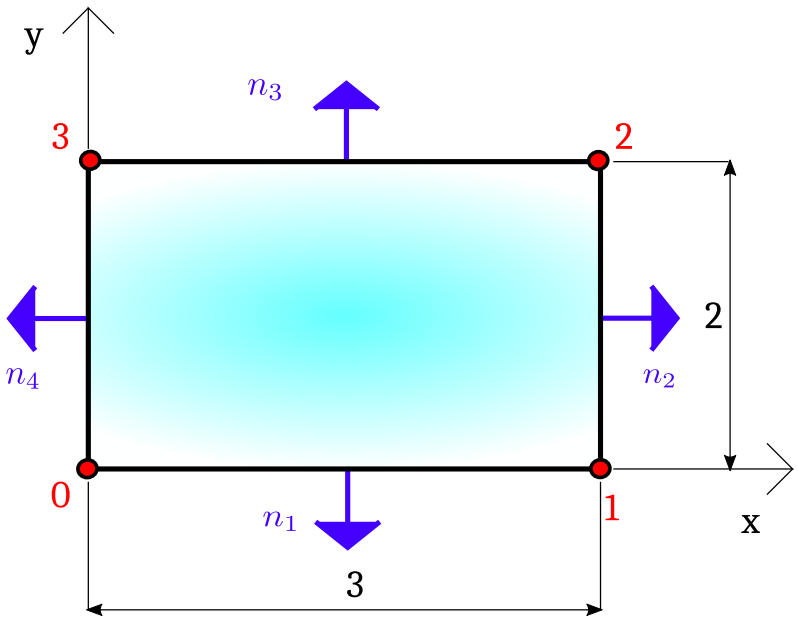


Figure 1.25: Normals of the elements

The coordinates of nodes from Fig. 1.25 may be written in matrix notation as

$$\mathbf{Coord} = \begin{matrix} \\ 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{matrix} \begin{matrix} x & y \end{matrix} \\ \begin{pmatrix} 0 & 0 \\ 3 & 0 \\ 3 & 2 \\ 0 & 2 \end{pmatrix} \end{matrix} . \tag{1.256}$$

The tangent vectors to elements gives the matrix **tg** , i.e.

$$\mathbf{tg} = \begin{pmatrix} 1-0 \\ 2-1 \\ 3-2 \\ 0-3 \end{pmatrix} = \begin{pmatrix} 3 & 0 \\ 0 & 2 \\ -3 & 0 \\ 0 & -2 \end{pmatrix} . \tag{1.257}$$

The tangents are not unit. The magnitude (norm or length) of the tangent vector can be calculated as

$$\|tg_i\| = \sqrt{tg_{xi}^2 + tg_{yi}^2} \tag{1.258}$$

where i denotes a position of tangent vector in the matrix **tg** and, for this example, is defined as $i = 1, 2, 3, 4$. It implies that

$$\|\mathbf{tg}\| = \begin{pmatrix} 3 \\ 2 \\ 3 \\ 2 \end{pmatrix} . \tag{1.259}$$

The unit tangent vector is determined as

$$tg_{u_i} = \frac{tg_i}{\|tg_i\|} \tag{1.260}$$

and from iterative process follows the matrix of unit tangent vectors $\mathbf{tg_u}$ that is

$$\mathbf{tg_u} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} . \tag{1.261}$$

Finally, the unit normal vector is defined by

$$n_i = (tg_{uy_i}, -tg_{ux_i}) \tag{1.262}$$

i.e.

$$\mathbf{n} = \begin{matrix} \\ \mathbf{n_1} \\ \mathbf{n_2} \\ \mathbf{n_3} \\ \mathbf{n_4} \end{matrix} \begin{matrix} \begin{matrix} n_x & n_y \end{matrix} \\ \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \end{pmatrix} \end{matrix} . \tag{1.263}$$

1.6 Comparison of FEM and BEM

The main advantages and disadvantages of both methods are introduced in Table 1.2

	FEM		**BEM**	
Discretisation	Full region	-	Boundary only	+
Stiffness matrix	Symmetrical	+	Non-symmetrical, Full	-
Solution	Faster, Sparse matrices	+	Full matrices	-
Non-linearities	Easier solution	+	More complicated	-
Implementation	Easier	+	More complicated	-

Table 1.2: Comparison of FEM and BEM

BEM is suitable for problems where the ratio between volume and surface is large. The derivation of the system of linear equations from differential equation based on the method of weighted residuals is described in Fig. 1.26.

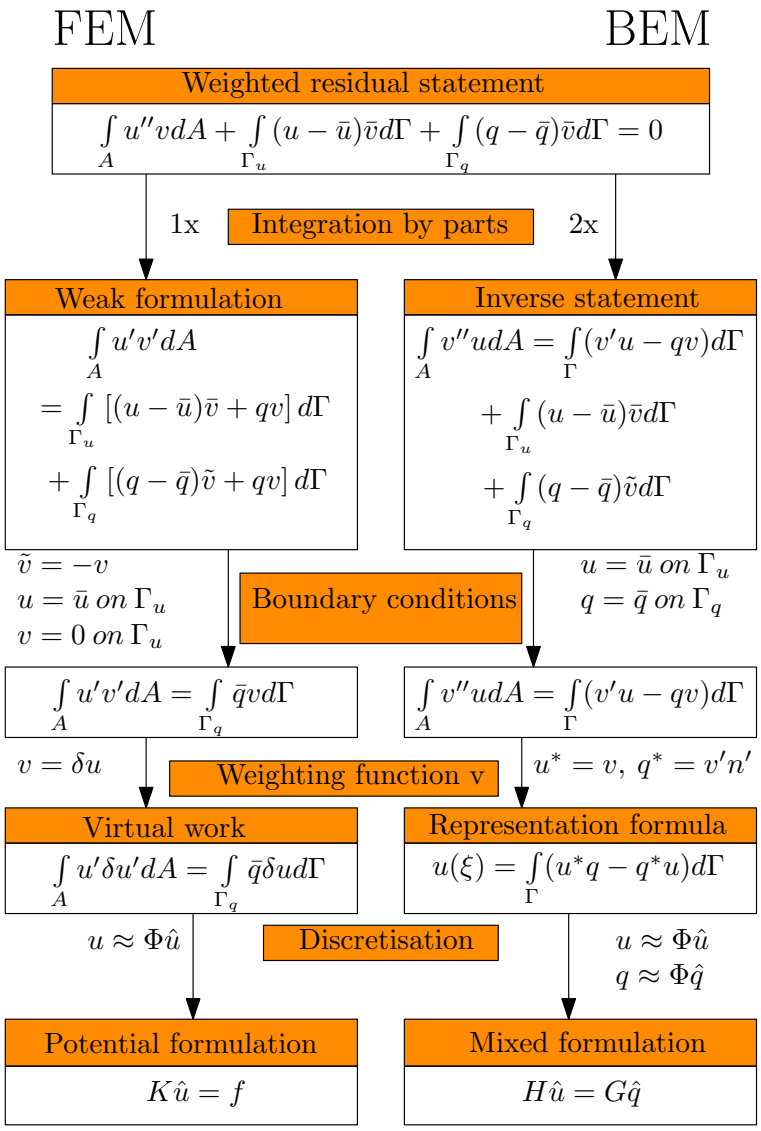


Figure 1.26: FEM vs BEM procedure[13]

Chapter 2

Numerical methods used for solving of torsion problems

The previous chapter was focused on the theory and it will be applied here in order to introduce the steps of the implementation. Before the application, it is convenient to introduce a derivation of simple one-dimensional problem to see the difference.

2.1 Illustration on simple 1D problem

A general notation of a one-dimensional differential equation in range $a \leq x \leq b$ may be expressed as

$$\mathcal{L}u(x) + g(x) = 0; \quad a \leq x \leq b \tag{2.1}$$

where $u(x)$ denotes unknown field (e.g. displacement, temperature, etc.) and $g(x)$ is known function (e.g. external load, heat supply,...) and $\mathcal{L}$ is called differential operator (e.g. $\mathcal{L} = \mathrm{d}/\mathrm{d}x$, $\mathcal{L} = \mathrm{d}^2/\mathrm{d}x^2 + 2$, and the like). Assume a differential equation

$$\frac{\mathrm{d}^2 u(x)}{\mathrm{d}x^2} + u = -x; \quad 0 \leq x \leq 1 \tag{2.2}$$

where x is one-dimensional domain. In accordance with Equation (2.1), the notation corresponds

$$\mathcal{L} = \frac{\mathrm{d}^2}{\mathrm{d}x^2} + 1; \quad g = x \ . \tag{2.3}$$

To obtain unique solution, it is essential to prescribe boundary conditions. They are given by

$$u(0) = u(1) = 0 \ . \tag{2.4}$$

In this case, it is feasible to acquire the analytical solution of the differential equation. The exact solution of (2.2) with boundary conditions (2.4) is equal to

$$u = \frac{\sin x}{\sin 1} - x \ . \tag{2.5}$$

Equation (2.2) will be solved numerically using FEM and BEM, and in the end, compared with the analytical solution.

2.1.1 FEM

The differential equation (2.2) is multiplied by integral and weight (test) function $v(x)$, i.e.

$$\int_0^1 v(x)\left(\frac{\mathrm{d}^2u(x)}{\mathrm{d}x^2}\mathrm{d}x + u + x\right)\mathrm{d}x = 0 \ . \tag{2.6}$$

From integration by parts, it follows that

$$\int_0^1 \frac{\mathrm{d}^2u(x)}{\mathrm{d}x^2}v(x)\mathrm{d}x = \left[\frac{\mathrm{d}u(x)}{\mathrm{d}x}v(x)\right]_0^1 - \int_0^1 \frac{\mathrm{d}u(x)}{\mathrm{d}x}\frac{\mathrm{d}v(x)}{\mathrm{d}x}\mathrm{d}x \tag{2.7}$$

and by inserting (2.7) into (2.6) it is obtained

$$\left[\frac{\mathrm{d}u(x)}{\mathrm{d}x}v(x)\right]_0^1 - \int_0^1 \frac{\mathrm{d}u(x)}{\mathrm{d}x}\frac{\mathrm{d}v(x)}{\mathrm{d}x}dx + \int_0^1 u(x)v(x)\mathrm{d}x + \int_0^1 xv(x)\mathrm{d}x = 0 \tag{2.8}$$

or

$$\int_0^1 \frac{\mathrm{d}u(x)}{\mathrm{d}x}\frac{\mathrm{d}v(x)}{\mathrm{d}x}\mathrm{d}x - \int_0^1 u(x)v(x)\mathrm{d}x = \left[\frac{\mathrm{d}u(x)}{\mathrm{d}x}v(x)\right]_0^1 + \int_0^1 xv(x)\mathrm{d}x \ . \tag{2.9}$$

The third term can be decomposed as

$$\int_0^1 \frac{\mathrm{d}u(x)}{\mathrm{d}x}\frac{\mathrm{d}v(x)}{\mathrm{d}x}\mathrm{d}x - \int_0^1 u(x)v(x)\mathrm{d}x = \left.\frac{\mathrm{d}u(x)}{\mathrm{d}x}\right|_{x=1} v(1) - \left.\frac{\mathrm{d}u(x)}{\mathrm{d}x}\right|_{x=0} v(0) + \int_0^1 xv(x)\mathrm{d}x \tag{2.10}$$

and using boundary conditions (2.4), (2.10) reduces to

$$\int_0^1 \frac{\mathrm{d}v(x)}{\mathrm{d}x}\frac{\mathrm{d}u(x)}{\mathrm{d}x}\mathrm{d}x - \int_0^1 v(x)u(x)\mathrm{d}x = \int_0^1 v(x)x\mathrm{d}x \ . \tag{2.11}$$

The unknown field $u(x)$ is replaced by an approximation, as shown in Fig. 2.1.

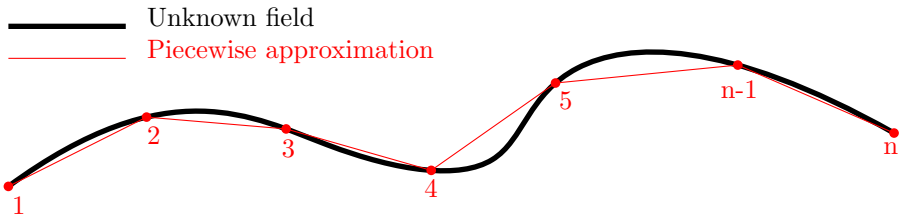


Figure 2.1: Exact function and approximation

The approximate function may be written as

$$u(x) = \mathbf{N}\hat{\mathbf{u}} \tag{2.12}$$

where

$$\mathbf{N} = \begin{pmatrix} N_1 & N_2 & \cdots & N_n \end{pmatrix}; \hat{\mathbf{u}} = \begin{pmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \vdots \\ \hat{u}_n \end{pmatrix} \tag{2.13}$$

where n corresponds to number of nodes. Note that the principle is the same as in Section 1.5.1d. According to Eq. (2.11), it is vital to have derivative of the unknown field. It implies that

$$\frac{\mathrm{d}u(x)}{\mathrm{d}x} = \mathbf{G}_F \hat{\mathbf{u}} \tag{2.14}$$

since the vector $\hat{\mathbf{u}}$ does not depend on the coordinate x. The matrix $\mathbf{G}_F$ is defined as

$$\mathbf{G}_F = \frac{\mathrm{d}\mathbf{N}}{\mathrm{d}x} = \begin{pmatrix} \frac{\mathrm{d}N_1}{\mathrm{d}x} & \frac{\mathrm{d}N_2}{\mathrm{d}x} & \dots & \frac{\mathrm{d}N_n}{\mathrm{d}x} \end{pmatrix} . \tag{2.15}$$

With (2.11), (2.12) and (2.14) follows that

$$\left(\int_0^1 \frac{\mathrm{d}v(x)}{\mathrm{d}x} \mathbf{G}_F \mathrm{d}x \right) \hat{\mathbf{u}} - \left(\int_0^1 v(x) \mathbf{N} \mathrm{d}x \right) \hat{\mathbf{u}} = \int_0^1 v(x) x \mathrm{d}x \tag{2.16}$$

It remains to choose the weight function $v(x)$. According to Galerkin method, the weight (test) function is the same as trial (basis) function, i.e.

$$v(x) = \mathbf{N}\ \mathbf{c} = \mathbf{c}^\top \mathbf{N}^\top \tag{2.17}$$

since v is a scalar. The derivative of the weight function is then

$$\frac{\mathrm{d}v(x)}{\mathrm{d}x} = \mathbf{c}^\top \mathbf{G}_F^\top \tag{2.18}$$

because v is arbitrary function, therefore, matrix $\mathbf{c}$ is also arbitrary. Combining (2.16), (2.17) and (2.18) implies

$$\mathbf{c}^T \left(\int_0^1 \mathbf{G}_F^\top \mathbf{G}_F \mathrm{d}x - \int_0^1 \mathbf{N}^\top \mathbf{N} \mathrm{d}x \right) \hat{\mathbf{u}} = \mathbf{c}^T \int_0^1 \mathbf{N}^\top x \mathrm{d}x \ . \tag{2.19}$$

Since the matrix $\mathbf{c}$ does not depend on x, Eq. (2.19) results in

$$\left[\int_0^1 \left(\mathbf{G}_F^\top \mathbf{G}_F \ - \mathbf{N}^\top \mathbf{N} \right) \mathrm{d}x \right] \hat{\mathbf{u}} = \int_0^1 \mathbf{N}^\top x \mathrm{d}x \tag{2.20}$$

and with matrix notation, Equation (2.20) takes form

$$\mathbf{K}\hat{\mathbf{u}} = \mathbf{f} \tag{2.21}$$

where

$$\mathbf{K} = \int_0^1 \left(\mathbf{G}_F^\top \mathbf{G}_F \ - \mathbf{N}^\top \mathbf{N} \right) \mathrm{d}x \tag{2.22}$$

denotes the stiffness matrix and

$$\mathbf{f} = \int_0^1 \mathbf{N}^\top x \mathrm{d}x \tag{2.23}$$

is called force vector. Establish the elements with linear shape function (see Fig. 1.24), where the element shape function matrix is defined as

$$\mathbf{N}^e = \begin{pmatrix} N_1 & N_2 \end{pmatrix} \tag{2.24}$$

where

$$\begin{aligned} N_1 &= 1 - \frac{x}{L^e} \\ N_2 &= \frac{x}{L^e} \end{aligned} \quad . \tag{2.25}$$

It is clear, that the derivative of the element shape function matrix gives

$$\mathbf{G}_F^e = \frac{\mathrm{d}\mathbf{N}^e}{\mathrm{d}x} = \begin{pmatrix} -\frac{1}{L^e} & \frac{1}{L^e} \end{pmatrix} \quad . \tag{2.26}$$

2.1.2 BEM

The first two steps are the same as in Section 2.1.1, let the author start the derivation from Equation (2.8), which is given by

$$\left[\frac{\mathrm{d}u(x)}{\mathrm{d}x}v(x)\right]_0^1 - \int_0^1 \frac{\mathrm{d}u(x)}{\mathrm{d}x}\frac{\mathrm{d}v(x)}{\mathrm{d}x}\mathrm{d}x + \int_0^1 u(x)v(x)\mathrm{d}x + \int_0^1 xv(x)\mathrm{d}x = 0 \ . \tag{2.27}$$

The second term can be integrated by parts again, i.e.

$$-\int_0^1 \frac{\mathrm{d}u(x)}{\mathrm{d}x}\frac{\mathrm{d}v(x)}{\mathrm{d}x}\mathrm{d}x = -\left[u(x)\frac{\mathrm{d}v(x)}{\mathrm{d}x}\right]_0^1 + \int_0^1 \frac{\mathrm{d}^2v(x)}{\mathrm{d}x^2}u(x)\mathrm{d}x \ . \tag{2.28}$$

Insertation of Eq. (2.28) into (2.27) gives

$$\int_0^1 \left(\frac{\mathrm{d}^2v(x)}{\mathrm{d}x^2} + v(x)\right)u(x)\mathrm{d}x + \int_0^1 xv(x)\mathrm{d}x + \left[\frac{\mathrm{d}u(x)}{\mathrm{d}x}v(x)\right]_0^1 - \left[u(x)\frac{\mathrm{d}v(x)}{\mathrm{d}x}\right]_0^1 = 0 \ . \tag{2.29}$$

The boundary terms in square brackets contain known (potentials are given by boundary conditions) and unknown (derivatives of the field) boundaries. It is used of Dirac's delta property (Read more in Appendix A) and a fundamental solution u^* as

$$\mathcal{L}^*u^* = \frac{\mathrm{d}^2u^*}{\mathrm{d}x^2} + u^* = \delta(x,\xi) \tag{2.30}$$

The fundamental solution represents an analytical solution of the governing differential equation for a point source [13]. ξ is a load point, where the point source is applied. If the fundamental solution u^* is defined as weight function $v(x)$, it gives with a sifting property of Dirac's delta (see Appendix A) for the first term from (2.29)

$$\int_0^1 \left(\frac{\mathrm{d}^2u^*}{\mathrm{d}x^2} + u^*\right)u\mathrm{d}x = \int_0^1 \delta(x,\xi)u(x)\mathrm{d}x = u(\xi) \ . \tag{2.31}$$

Insertation of (2.31) in (2.29) yields

$$u(\xi) = \int_0^1 xu^*(x,\xi)\mathrm{d}x + \left[\frac{\mathrm{d}u}{\mathrm{d}x}u^*\right]_0^1 - \left[u\frac{\mathrm{d}u^*}{\mathrm{d}x}\right]_0^1 \tag{2.32}$$

which is called representation formula. The fundamental solution is determined from Helmholtz equation, i.e.

$$u^* = \frac{\sin|x-\xi|}{2} \ . \tag{2.33}$$

Combining Eqs. (2.32),(2.4) and (2.33), it is obtained the representation formula

$$\begin{aligned} u(\xi) &= -\int_0^1 \frac{x}{2}\sin|x-\xi|\mathrm{d}x - \left[u(x)'\frac{1}{2}\sin|x-\xi|\right]_0^1 \\ &= -\int_0^1 \frac{x}{2}\sin|x-\xi|\mathrm{d}x - \left(\frac{u(1)'}{2}\sin|x-1| - \frac{u(0)'}{2}\sin|x|\right) \ . \end{aligned} \tag{2.34}$$

The load point can be placed on boundaries, i.e. $u(\xi=0)$, $u(\xi=1)$ and it gives two equations from (2.34). The unknown derivatives of potential $u(0)'$ and $u(1)'$ may be calculated. Equations may be written in matrix notation as

$$\begin{pmatrix} -\frac{\sin 1}{2} & \frac{\sin 0}{2} \\ -\frac{\sin 0}{2} & \frac{\sin 1}{2} \end{pmatrix} \begin{pmatrix} u(1)' \\ u(0)' \end{pmatrix} = \begin{pmatrix} \int_0^1 \frac{x}{2}\sin|x|\mathrm{d}x + u(0) \\ \int_0^1 \frac{x}{2}\sin|x-1|\mathrm{d}x + u(1) \end{pmatrix}$$

and the matrix solution results in

$$u(0)' = \frac{1}{\sin 1} - 1 \tag{2.35}$$

and

$$u(1)' = \frac{\cos 1}{\sin 1} - 1 \ . \tag{2.36}$$

With (2.35) and (2.36), (2.34) reduces to

$$u(\xi) = -\frac{1}{2}\left(\frac{\cos 1}{\sin 1} - 1\right)\sin|1-\xi| + \frac{1}{2}\left(\frac{1}{\sin 1} - 1\right)\sin\xi - \int_0^1 \frac{x}{2}\sin|x-\xi|\mathrm{d}x \tag{2.37}$$

where is possible to compute any arbitrary value inside interval $\xi \in (0,1)$. The domain was divided into five finite elements and values of the potential u were calculated for nine inner nodes from BEM as shown in Figure (2.2). Results from FEM are exact in nodes as well as BEM results.

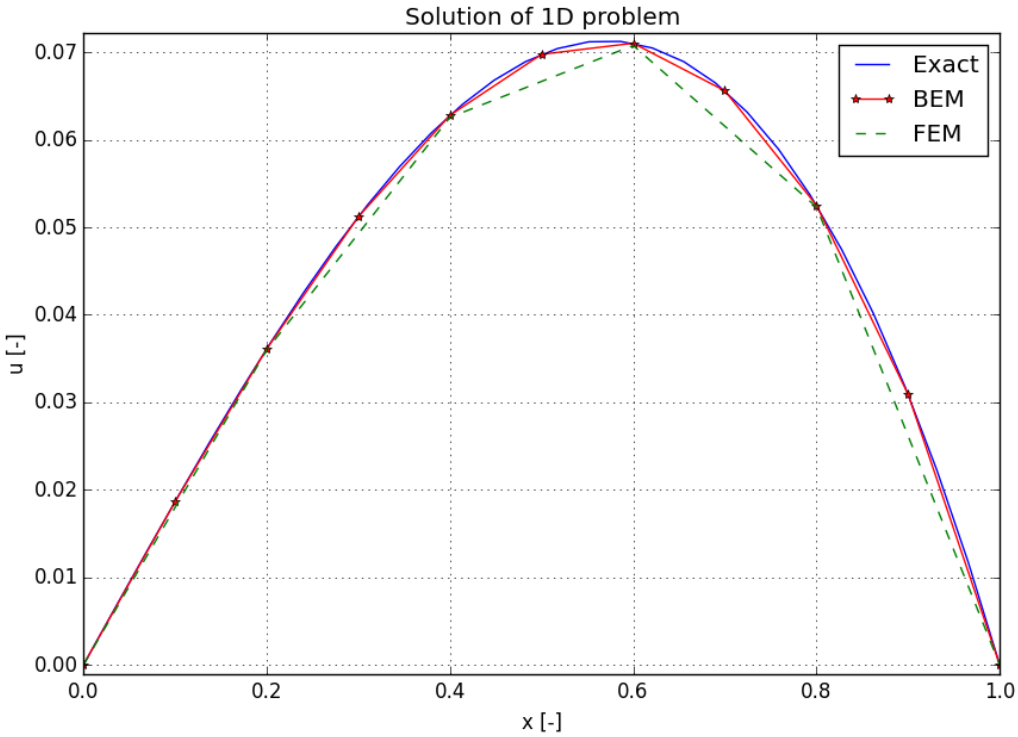


Figure 2.2: Comparison of exact and numerical solution

2.2 Introduction of the problem

It is considered a shaft (see Fig. 1) with square cross-section, as shown in Figure 2.3. The length is 1 m and the side of the square measures 0.1 m. The shaft is made of steel with Young's modulus E = $2.1{\cdot}10^5$ MPa and Poisson ratio $\mu = 0.3$. The value of the rate of twist is 0.01 rad/m. The shear modulus can be calculated according to (1.5), i.e.

$$G = \frac{E}{2(1+\mu)} = \frac{2.1 \cdot 10^5}{2(1+0.3)} \doteq 80\,769 MPa \ . \tag{2.38}$$

The task is to compute and compare the value of the displacement of the membrane over the cross-section.

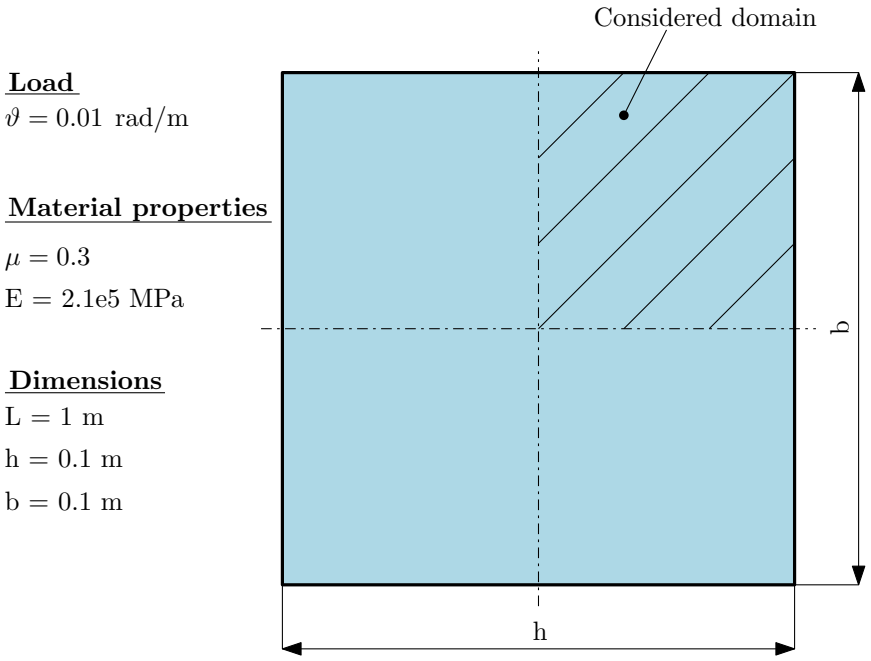


Figure 2.3: 2D example

2.3 FEM

The cross-section 1 is symmetrical, therefore only one-fourth of the cross-section may be analyzed. The boundary conditions will be described with this consideration. The analyzed part is divided into four elements with five nodes (degrees of freedom) as shown in Fig. 2.4.

Coordinates of nodes can be arranged into matrix as

$$\mathbf{coordinates} = \begin{matrix} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{matrix} x \\ \begin{pmatrix} 0.000 \\ 0.050 \\ 0.025 \\ 0.050 \\ 0.000 \end{pmatrix} \end{matrix} \begin{matrix} y \\ \begin{pmatrix} 0.000 \\ 0.000 \\ 0.025 \\ 0.050 \\ 0.050 \end{pmatrix} \end{matrix} . \tag{2.39}$$

Elements consist of nodes and the matrix may be defined as

$$\mathbf{elements} = \begin{matrix} ⓪ \\ ① \\ ② \\ ③ \end{matrix} \begin{pmatrix} 0 & 1 & 2 \\ 1 & 3 & 2 \\ 3 & 4 & 2 \\ 4 & 0 & 2 \end{pmatrix} . \tag{2.40}$$

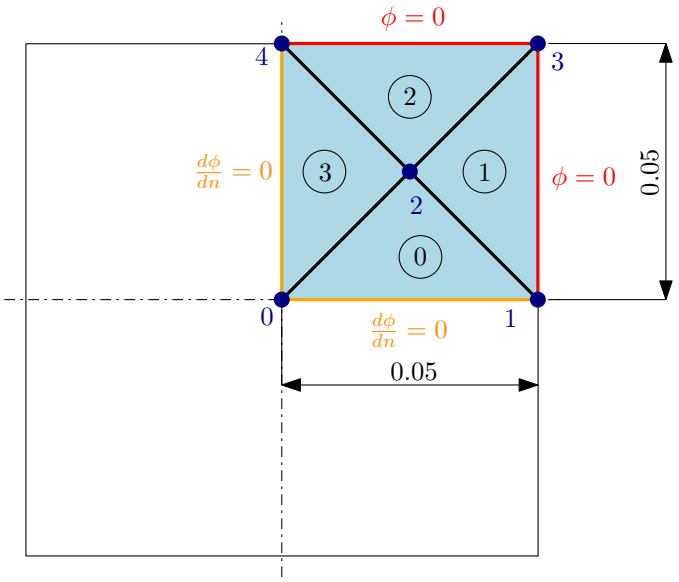


Figure 2.4: 2D example - FEM

If the Dirichlet boundary conditions is applied, index is true = 1. Otherwise index is zero = 0. The array as given as

$$\textbf{isOnEdge} = \Big(0, 1, 0, 1, 1\Big) \ . \tag{2.41}$$

The interpretation of **isOnEdge** is that in nodes 1,3 and 4 are applied Dirichlet boundary condition, which causes zero displacement. The matrices are established due to an iterative process. The number of iterations depends on number of elements. For each element, it is calculated the stiffness matrix and the load vector. According to (1.128), it is possible to write an element stiffness matrix as

$$\mathbf{K}^e = \iint_{A^e} \mathbf{G}_F^{e\top} \mathbf{D} \mathbf{G}_F^e \mathrm{d}A \ . \tag{2.42}$$

Since both $\mathbf{G}_F^e$ (1.153) and $\mathbf{D}$ (1.68) do not depend on coordinates x and y, the double integral can be expressed as area A^e of the element. With (1.68), (2.42) results in

$$\mathbf{K}^e = \frac{A^e}{G} \mathbf{G}_F^{e\top} \mathbf{G}_F^e \ . \tag{2.43}$$

The area of the element A^e is the same for all triangles and can be calculated as

$$A^e = \frac{0.05^2}{4} = 6.25 \cdot 10^{-4} \tag{2.44}$$

The derivative of the element shape function matrix gives in accordance with (1.153) for the first element

$$\mathbf{G}_F^e = \frac{1}{2 \cdot 6.25 \cdot 10^{-4}} \begin{pmatrix} 0 - 0.025 & 0.025 - 0 & 0 - 0 \\ 0.025 - 0.05 & 0 - 0.025 & 0.05 - 0 \end{pmatrix} = \begin{pmatrix} -20 & 20 & 0 \\ -20 & -20 & 40 \end{pmatrix} \ . \tag{2.45}$$

Combining (2.38), (2.43) and (2.45), it is concluded that

$$\mathbf{K}^e = \frac{6.25 \cdot 10^{-4}}{8.0769 \cdot 10^{10}} \begin{pmatrix} -20 & -20 \\ 20 & -0 \\ 0 & 40 \end{pmatrix} \begin{pmatrix} -20 & 20 & 0 \\ -20 & -20 & 40 \end{pmatrix}$$

$$= 10^{-11} \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0.628 & 0 & -0.62 \\ 0 & 0.62 & -0.62 \\ -0.62 & -0.62 & 1.24 \end{pmatrix} \end{array} \tag{2.46}$$

where the blue numbers correspond to nodes. In accordance with this numbering, it is inserted the contribution to the global stiffness matrix (see (2.51)). The element load vector may be calculated in compliance with (1.130), i.e.

$$\mathbf{f_l}^e = 2\vartheta_1 \iint_{A^e} \mathbf{N}^{e\top} \, \mathrm{d}A \ . \tag{2.47}$$

According to Figure 1.14, each shape function varies linearly. Therefore, the integral may be assessed very simply from the volume under each shape function, which is defined by

$$V = \frac{1}{3} h \cdot A \tag{2.48}$$

where h is the height (equal to 1) and A denotes the area of the element. Thus, the element load vector becomes

$$\mathbf{f_l}^e = \frac{2}{3} \vartheta A^e \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.49}$$

and the numerical evaluation is defined as

$$\mathbf{f_l}^e = \frac{2}{3} 0.01 \cdot 6.25 \cdot 10^{-4} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 2.16667 \cdot 10^{-6} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \ . \tag{2.50}$$

The global system of equations is given by

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1.24 & 0 & -1.24 & 0 & 0 \\ 0 & 1.24 & -1.24 & 0 & 0 \\ -1.24 & -1.24 & 4.95 & -1.24 & -1.24 \\ 0 & 0 & -1.24 & 1.24 & 0 \\ 0 & 0 & -1.24 & 0 & 1.24 \end{pmatrix} \end{array} 10^{-11} \begin{pmatrix} \phi_0 \\ 0 \\ \phi_2 \\ 0 \\ 0 \end{pmatrix} = 10^{-6} \begin{pmatrix} 8.33 \\ 8.33 \\ 16.67 \\ 8.33 \\ 8.33 \end{pmatrix} \tag{2.51}$$

and with application of boundary conditions,

$$\begin{array}{c} \\ 0 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 2 \end{array} \\ \begin{pmatrix} 1.24 & -1.24 \\ -1.24 & 4.95 \end{pmatrix} \end{array} 10^{-11} \begin{pmatrix} \phi_0 \\ \phi_2 \end{pmatrix} = 10^{-6} \begin{pmatrix} 8.33 \\ 16.67 \end{pmatrix} \ . \tag{2.52}$$

Then, the solution results in

$$\begin{pmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix} = \begin{pmatrix} 1232751 \\ 0 \\ 616375 \\ 0 \\ 0 \end{pmatrix} . \tag{2.53}$$

The displacement ϕ_2 is equal to ϕ_{FEM} and will be compared with displacement obtained from BEM. The stiffness matrix is symmetrical as a consequence of the Galerkin method. Non-zero values (blue color) of the global stiffness matrix are shown in Fig. 2.5 and the reader can compare the matrix with Fig. 1.15.

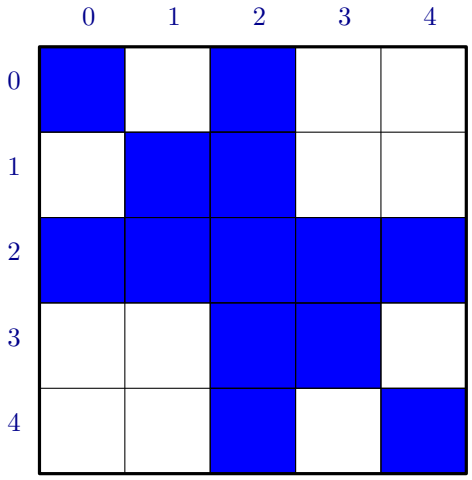


Figure 2.5: The stiffness matrix

2.4 BEM

The BE mesh is created from FE nodes (2.4), where the FE nodes remain unchanged. The light blue domain in Fig. 2.6 is transformed into boundary (orange and red lines). One boundary element is created from two original nodes from FEM and node of boundary element is located between them (constant element shape function). The domain possess one inner node (blue-yellow), which has the same coordinates as node 2 from Fig. 2.4. The matrix of coordinates is given by

$$\mathbf{Coord} = \begin{matrix} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{matrix} \begin{matrix} x & \end{matrix} \\ \begin{pmatrix} 0.000 & 0.000 \\ 0.050 & 0.000 \\ 0.025 & 0.025 \\ 0.050 & 0.050 \\ 0.000 & 0.050 \end{pmatrix} \end{matrix} \begin{matrix} \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} y \\ \\ \\ \\ \\ \end{matrix} . \tag{2.54}$$

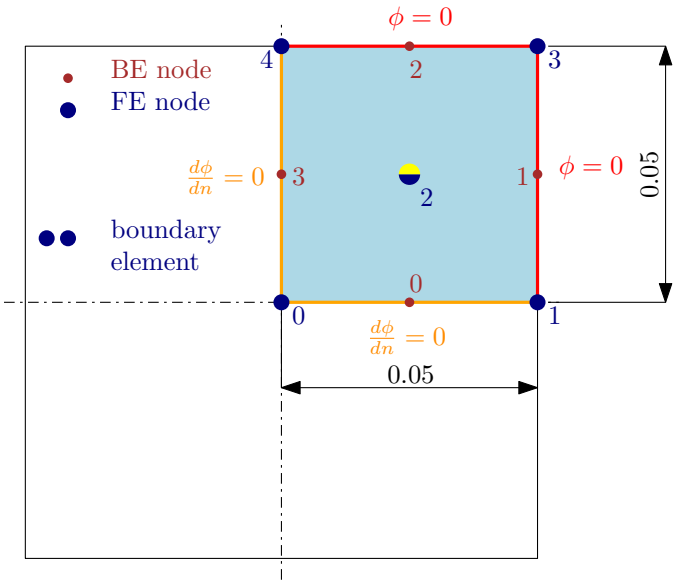


Figure 2.6: 2D example - BEM

The normals are computed in compliance with Section 1.5.1g and the matrix results in

$$\mathbf{n} = \begin{matrix} \\ \mathbf{n_0} \\ \mathbf{n_1} \\ \mathbf{n_2} \\ \mathbf{n_3} \end{matrix} \begin{matrix} \begin{matrix} n_x & n_y \end{matrix} \\ \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 0 & 1 \\ -1 & 0 \end{pmatrix} \end{matrix} . \tag{2.55}$$

The boundary nodes are located between elements and they can be arranged into matrix as

$$\mathbf{coor_{nod}} = \begin{matrix} \\ 0 \\ 1 \\ 2 \\ 3 \end{matrix} \begin{matrix} \begin{matrix} x & y \end{matrix} \\ \begin{pmatrix} 0.025 & 0.000 \\ 0.050 & 0.025 \\ 0.025 & 0.050 \\ 0.000 & 0.025 \end{pmatrix} \end{matrix} . \tag{2.56}$$

Two Gauss integration points (see Appendix C) for interval (0,1) are given as

$$\boldsymbol{\omega} = \begin{pmatrix} 0.21132486540518713 \\ 0.7886751345948129 \end{pmatrix} \tag{2.57}$$

$$\mathbf{w} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} . \tag{2.58}$$

If the node contains Dirichlet boundary condition, index of the matrix **BCu** is equal to 1. Otherwise Neumann BC is applied (index is zero), i.e.

$$\mathbf{BCu} = \begin{pmatrix} 0, 1, 1, 0 \end{pmatrix} . \tag{2.59}$$

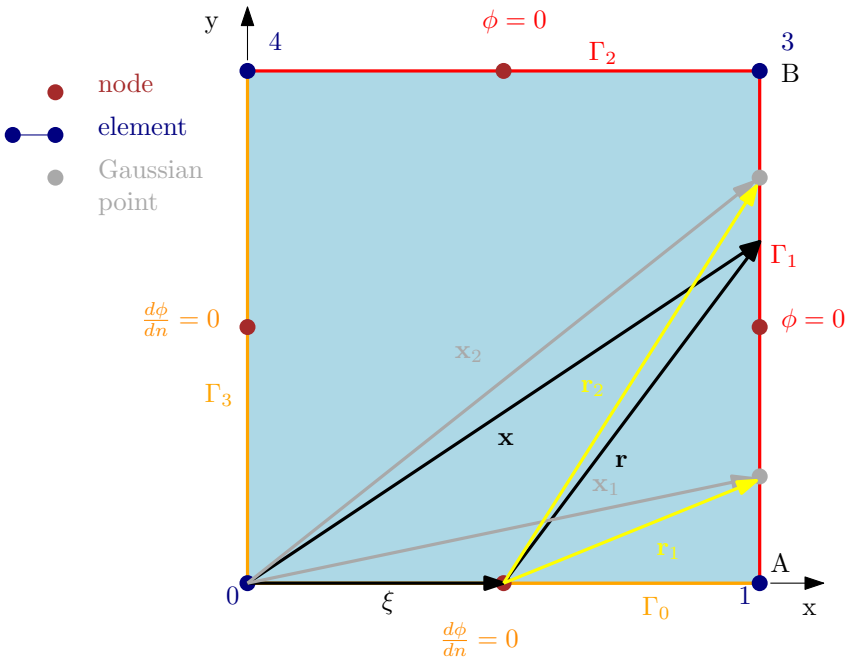


Figure 2.7: BEM vectors

The system of linear equations is according to (1.248)

$$\begin{pmatrix} H_{00} & H_{01} & H_{02} & H_{03} \\ H_{10} & H_{11} & H_{12} & H_{13} \\ H_{20} & H_{21} & H_{22} & H_{23} \\ H_{30} & H_{31} & H_{32} & H_{33} \end{pmatrix} \begin{pmatrix} \hat{u}_0 \\ \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \end{pmatrix} = \begin{pmatrix} G_{00} & G_{01} & G_{02} & G_{03} \\ G_{10} & G_{11} & G_{12} & G_{13} \\ G_{20} & G_{21} & G_{22} & G_{23} \\ G_{30} & G_{31} & G_{32} & G_{33} \end{pmatrix} \begin{pmatrix} \hat{q}_0 \\ \hat{q}_1 \\ \hat{q}_2 \\ \hat{q}_3 \end{pmatrix} . \tag{2.60}$$

Suppose a calculation of the matrix components H_{01} and G_{01}. The load point is collocated in node 0 and the boundary Γ_1, as shown in Fig. 2.7. Employing Gaussian quadrature on Eq. (1.245) and using Eqs. (1.217) and (1.218) imply for two Gaussian point

$$G_{01} = \sum_{i=1}^{2} w_i \cdot \frac{\ln \|\mathbf{r}\|}{-2\pi} \cdot len_{ab} \tag{2.61}$$

$$H_{01} = \sum_{i=1}^{2} w_i \cdot \frac{\mathbf{r} \cdot \mathbf{n}}{-2\pi |\mathbf{r}|^2} \cdot len_{ab} \ . \tag{2.62}$$

The coordinate of the point A from Fig. 2.7 is [0.05,0] and point B = [0.05,0.05]. The coordinates may be written as an arrays

$$\mathbf{a} = \Big(0.05, 0\Big) \tag{2.63}$$

$$\mathbf{b} = \Big(0.05, 0.05\Big) \tag{2.64}$$

and the length between points A and B (Γ_1) is written as a norm

$$len_{ab} = \|\mathbf{a} - \mathbf{b}\| = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} \tag{2.65}$$

$$len_{ab} = 0.05 \tag{2.66}$$

The integration is carried out Gaussian quadrature for two Gaussian points (gray points in Fig. 2.7). It means that continuous vector $\mathbf{x}$ is replaced by two discrete vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, i.e.

$$\mathbf{x}_n = \sum_{i=1}^{2} \mathbf{a} + (\mathbf{b} - \mathbf{a})\omega_i \tag{2.67}$$

where ω_i denotes Gaussian point. In the same manner, the vector $\mathbf{r}$ is also discretised as

$$\mathbf{r}_n = \mathbf{x}_n - \boldsymbol{\xi} \tag{2.68}$$

Establish an iterative process that will compute the components of $\mathbf{H}$ and $\mathbf{G}$ matrices. The first iteration is given as

$$\mathbf{x}_1 = \mathbf{a} + (\mathbf{b} - \mathbf{a})\omega_1 \tag{2.69}$$

$$\mathbf{x}_1 = (0.05, 0) + (0.05 - 0.05, 0.05 - 0) \cdot 0.211325 = (0.05, 0.010566) \tag{2.70}$$

$$\mathbf{r}_1 = \mathbf{x}_1 - \boldsymbol{\xi} = (0.05, 0.010566) - (0.025, 0) = (0.025, 0.010566) \tag{2.71}$$

$$G_{01}^1 = w_1 \cdot \ln \|\mathbf{r}_1\| \cdot len_{ab} \tag{2.72}$$

$$G_{01}^1 = 0.5 \cdot (-3.6067) \cdot 0.05 = -0.090168 \tag{2.73}$$

$$H_{01}^1 = w_1 \cdot \frac{\mathbf{r}_1 \cdot \mathbf{n}_1}{|\mathbf{r}_1|^2} \cdot len_{ab} \tag{2.74}$$

$$H_{01}^1 = 0.5 \cdot \frac{0.025 \cdot 1 + 0.010566 \cdot 0}{|0.0271|^2} \cdot 0.05 = 0.8484 \tag{2.75}$$

and the second iteration is defined by

$$\mathbf{x}_2 = \mathbf{a} + (\mathbf{b} - \mathbf{a})\omega_2 \tag{2.76}$$

$$\mathbf{x}_2 = (0.05, 0) + (0.05 - 0.05, 0.05 - 0) \cdot 0.788675 = (0.05, 0.0.03943375) \tag{2.77}$$

$$\mathbf{r}_2 = \mathbf{x}_2 - \boldsymbol{\xi} = (0.05, 0.0.03943375) - (0.025, 0) = (0.025, 0.03943375) \tag{2.78}$$

$$G_{01}^2 = w_2 \cdot \ln \|\mathbf{r}_2\| \cdot len_{ab} \tag{2.79}$$

$$G_{01}^2 = 0.5 \cdot (-3.0642) \cdot 0.05 = -0.076605 \tag{2.80}$$

$$H_{01}^2 = w_2 \cdot \frac{\mathbf{r}_2 \cdot \mathbf{n}_1}{|\mathbf{r}_2|^2} \cdot len_{ab} \tag{2.81}$$

$$H_{01}^2 = 0.5 \cdot \frac{0.025 \cdot 1 + 0.03943375 \cdot 0}{|0.04669|^2} \cdot 0.05 = 0.28669 \ . \tag{2.82}$$

The components of matrices from iterations are added up and divided by constants. This results in

$$G_{01} = \frac{G_{01}^1 + G_{01}^2}{-2\pi} = 0.02654 \tag{2.83}$$

$$H_{01} = \frac{H_{01}^1 + H_{01}^2}{-2\pi} = -0.1807 \tag{2.84}$$

The diagonal elements of the matrix $\mathbf{G}$ contain singularity, however, the integrals exist and may be integrated analytically [32] as

$$G_{ii} = \frac{1 + \ln(2) - \ln \| \, len_{ab}\|}{2\pi} \cdot \| \, len_{ab}\| \tag{2.85}$$

The diagonal elements of matrix $\mathbf{H}$ vanish due to orthogonality between normal vector $\mathbf{n}$ and vector $\mathbf{r}$ (see Eq. (1.243)). Boundary conditions are according to Eq. (1.200) given as

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} \dfrac{0.05^2 + 0.025^2}{4} \\ \dfrac{0.025^2 + 0.05^2}{4} \end{pmatrix} = \begin{pmatrix} 7.8125 \cdot 10^{-4} \\ 7.8125 \cdot 10^{-4} \end{pmatrix} \quad . \tag{2.86}$$

If all matrix components are calculated, the matrices $\mathbf{H}$ and $\mathbf{G}$ may be assembled, i.e.

$$\begin{pmatrix} 0.5 & -0.181 & -0.147 & -0.181 \\ -0.181 & 0.5 & -0.181 & -0.147 \\ -0.147 & -0.181 & 0.5 & -0.181 \\ -0.181 & -0.147 & -0.181 & 0.5 \end{pmatrix} \begin{pmatrix} \hat{u}_0 \\ 7.8125 \cdot 10^{-4} \\ 7.8125 \cdot 10^{-4} \\ \tilde{u}_3 \end{pmatrix} = \begin{pmatrix} 0.037 & 0.027 & 0.024 & 0.027 \\ 0.027 & 0.037 & 0.027 & 0.024 \\ 0.024 & 0.027 & 0.037 & 0.027 \\ 0.027 & 0.024 & 0.027 & 0.037 \end{pmatrix} \begin{pmatrix} 0 \\ \hat{q}_1 \\ \hat{q}_2 \\ 0 \end{pmatrix} \quad . \tag{2.87}$$

The known variables may be separated to right-hand side and the system of matrices is rewritten as

$$\begin{pmatrix} 0.5 & -0.027 & -0.024 & -0.181 \\ -0.181 & -0.037 & -0.027 & -0.147 \\ -0.147 & -0.027 & -0.037 & -0.181 \\ -0.181 & -0.024 & -0.027 & 0.5 \end{pmatrix} \begin{pmatrix} \hat{u}_0 \\ \hat{q}_1 \\ \hat{q}_2 \\ \tilde{u}_3 \end{pmatrix} = \begin{pmatrix} 0.037 & 0.181 & 0.147 & 0.027 \\ 0.027 & -0.5 & 0.181 & 0.024 \\ 0.024 & 0.181 & -0.5 & 0.027 \\ 0.027 & 0.147 & 0.181 & 0.037 \end{pmatrix} \begin{pmatrix} 0 \\ 7.8125 \cdot 10^{-4} \\ 7.8125 \cdot 10^{-4} \\ 0 \end{pmatrix} \quad . \tag{2.88}$$

The solution of Eq. (2.88) leads to

$$\begin{pmatrix} \hat{u}_0 \\ \hat{q}_1 \\ \hat{q}_2 \\ \tilde{u}_3 \end{pmatrix} = \begin{pmatrix} 7.837 \cdot 10^{-4} \\ -1.132 \cdot 10^{-4} \\ -1.132 \cdot 10^{-4} \\ 7.837 \cdot 10^{-4} \end{pmatrix} \quad . \tag{2.89}$$

The resulting vectors of the original system from Eq. (2.87) are given by

$$\mathbf{u} = \begin{pmatrix} 7.837 \cdot 10^{-4} \\ 7.813 \cdot 10^{-4} \\ 7.813 \cdot 10^{-4} \\ 7.837 \cdot 10^{-4} \end{pmatrix} \; ; \; \mathbf{q} = \begin{pmatrix} 0 \\ -1.132 \cdot 10^{-4} \\ -1.132 \cdot 10^{-4} \\ 0 \end{pmatrix} \quad . \tag{2.90}$$

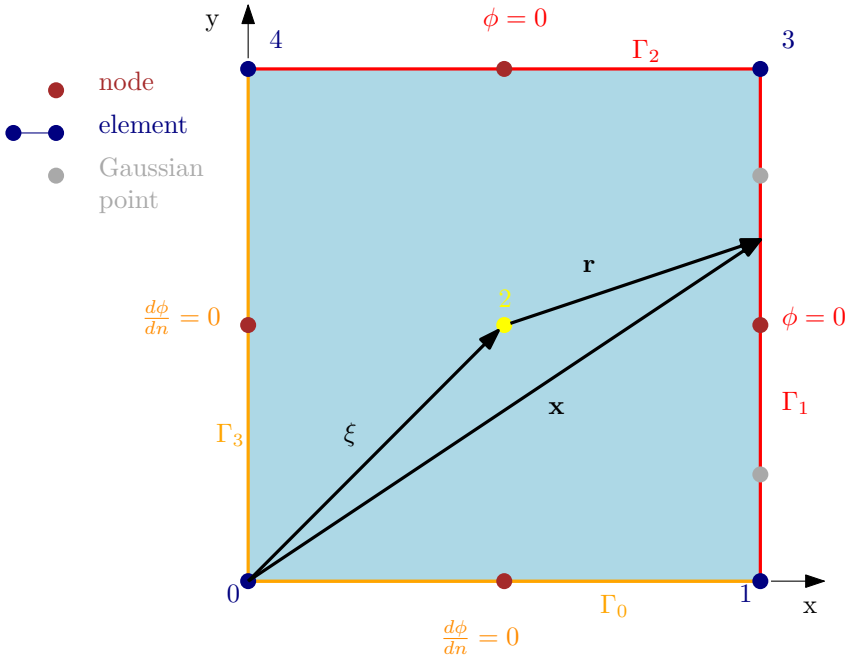


Figure 2.8: BEM vectors

According to Eq. (1.254), the components of matrices $\mathbf{H}_A$ and $\mathbf{G}_A$ may be computed in the same manner as components H_{01} and G_{01}. The load point is placed into point 2 (see Fig. 2.8) and from this point is carried out the integration of Γ_0 through Γ_3. It results in

$$\mathbf{H}_A = \begin{pmatrix} -0.2387 \\ -0.2387 \\ -0.2387 \\ -0.2387 \end{pmatrix} ; \quad \mathbf{G}_A = \begin{pmatrix} 0.0282 \\ 0.0282 \\ 0.0282 \\ 0.0282 \end{pmatrix} . \tag{2.91}$$

The displacement for unit right-hand side of the differential equation (1.187) may be calculated using Eq. (1.255) as

$$\tilde{u}(\xi_A) = \mathbf{G}_A^\top \hat{\mathbf{q}} - \mathbf{H}_A^\top \hat{\mathbf{u}} - \frac{1}{4}\left(x_A^2 + y_A^2\right) \tag{2.92}$$

$$\phi_{BEM} = \tilde{u}(\xi_A) 2G\vartheta \tag{2.93}$$

$$\phi_{BEM} = 4.28 \cdot 10^{-4} \cdot 2 \cdot 80\,769 \cdot 10^6 \cdot 0.01 = 691879 \ . \tag{2.94}$$

The difference between displacement from FEM and BEM can be calculated as

$$\frac{\phi_{BEM} - \phi_{FEM}}{\phi_{BEM}} = \frac{691879 - 616375}{691879} = 0.109 \ . \tag{2.95}$$

The difference is approximately 11%.

Chapter 3

Numerical analysis of chosen cross-sections using FEM and BEM

3.1 Introduction and analytical results for basic cross-sections

The analytical or semi-analytical solutions exist for basic cross-section, such as circle, rectangle, equilateral triangle, ellipse, etc. The formulas are written in [18], [3],[23] or [19]. It will be introduced results of analytical solution for circle, rectangle, equilateral triangle and right-angled triangle. This cross-sections were implemented into the mesh generator (see below).

3.1.1 Circle

The formulas for the shear stress and for the rate of twist are derived in 1.2. The equation for the maximum shear stess is equal to

$$\tau_{max} = \frac{T}{\frac{\pi}{16}d^3} \tag{3.1}$$

and the rate of twist is calculated as

$$\vartheta = \frac{T}{GI_p} \ . \tag{3.2}$$

3.1.2 Rectangle

Consider the rectangle as shown in Fig. 3.1, where the ratio h:b $>$ 1. The maximum stress is defined as

$$\tau_{max} = \frac{T}{\kappa_2 h b^2} \tag{3.3}$$

and the rate of twist is calculated as

$$\vartheta = \frac{T}{G\kappa_3 b^3 h} \ . \tag{3.4}$$

where the coefficients κ_1, κ_2 and κ_3 are written in Table 3.1. The table and formulas are taken from [19].

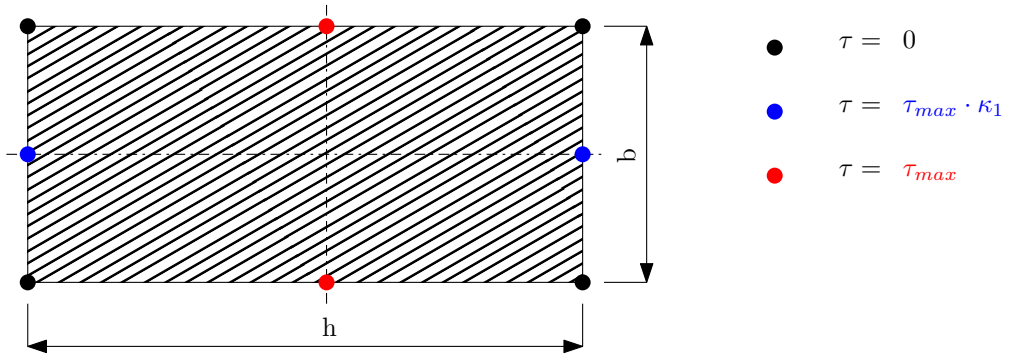


Figure 3.1: Rectangular cross-section

$\frac{h}{b}$	**1**	**1.2**	**1.5**	**2**	**3**	**5**	**10**	**∞**
κ_1	1	0.933	0.857	0.793	0.754	0.743	0.743	0.7435
κ_2	0.208	0.219	0.231	0.246	0.267	0.291	0.313	0.333
κ_3	0.1404	0.1661	0.1957	0.2286	0.2633	0.2808	0.3123	0.333

Table 3.1: Coefficients

3.1.3 Equilateral triangle

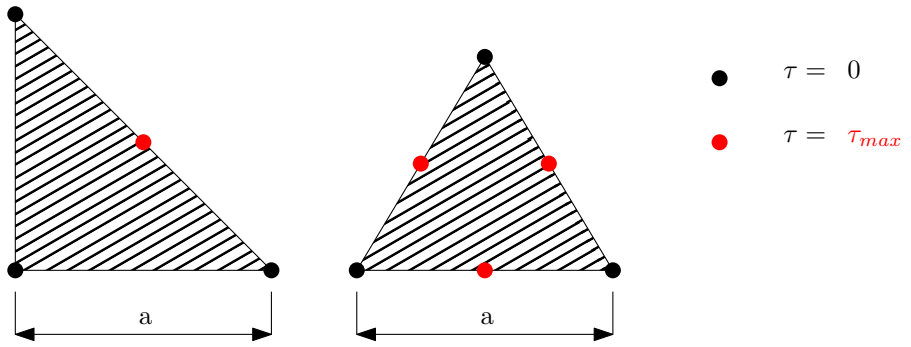


Figure 3.2: Triangular cross-sections

The formula for the maximum shear stress corresponds

$$\tau_{max} = \frac{20T}{a^3} \tag{3.5}$$

where a is the side of the triangle. The maximum stress is located in the middle of sides (see Fig. 3.2). The rate of twist is defined by

$$\vartheta = \frac{80T}{\sqrt{3}Ga^4} \ . \tag{3.6}$$

3.1.4 Right-angled triangle

The maximum stress is computed as

$$\tau_{max} = \frac{T}{0.05704a^3} \tag{3.7}$$

and it is placed in the middle of hypotenuse, as shown in Fig. 3.2. The rate of twist is equal to

$$\vartheta = \frac{T}{0.0260958Ga^4} \ . \tag{3.8}$$

The formulas for triangles are taken from [3].

3.1.5 Membrane analogy - numerical polar moment of inertia

The objective is to calculate the angle of twist by membrane analogy. The beam is loaded by torque T as shown in Fig. (3.3).

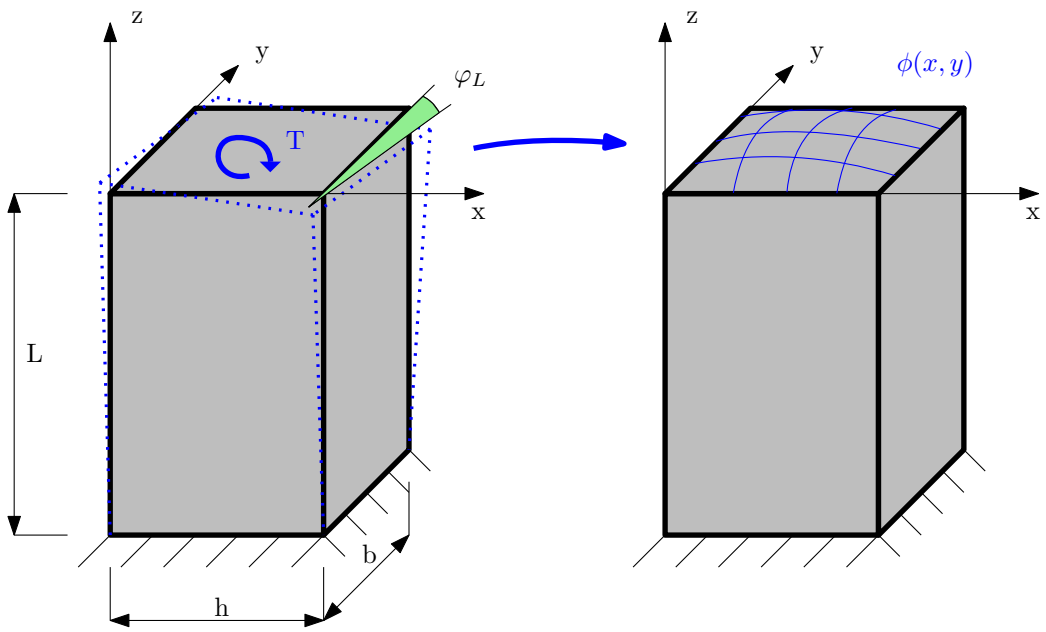


Figure 3.3: Beam

As derived before, the torque is equal to double of the volume under membrane (Prandtl's stress function). With (1.91) and (1.93) implies that

$$T = 2\iint_A \phi(x,y)\mathrm{d}A = 2\iint_A \phi(x,y)\mathrm{d}x\mathrm{d}y = 2V_\phi \ . \tag{3.9}$$

Recall the differencial equation (1.65)

$$\Delta\phi(x,y) = -2G\vartheta \tag{3.10}$$

and similar equation with unit right hand side (1.187)

$$\Delta u(x,y) = -1 \ . \tag{3.11}$$

It follows the relation between two variables, as shown in Equation (1.189). A double integral of the variable $u(x,y)$ is equal to the volume under membrane, i.e.

$$\iint_A u(x,y)\mathrm{d}A = \iint_A u(x,y)\mathrm{d}x\mathrm{d}y = V_u \ . \tag{3.12}$$

It is clear that

$$\iint_A \phi(x,y)\mathrm{d}A = \iint_A \phi(x,y)\mathrm{d}x\mathrm{d}y = 2G\vartheta V_u \ . \tag{3.13}$$

Using Eq. (3.13) in (3.9) results in

$$T = 2 \cdot 2G\vartheta V_u = 4G\vartheta V_u \ . \tag{3.14}$$

Remember Eq. (1.54)

$$\varphi_L = \vartheta L \tag{3.15}$$

and combining with (3.14) follows that

$$\varphi_L = \frac{TL}{4V_u G} \ . \tag{3.16}$$

Let relation $4V_u$ be defined as $I_{p'}$ and it is concluded that

$$\varphi_L = \frac{TL}{I_{p'} G} \ . \tag{3.17}$$

The quantity $I_{p'}$ corresponds according Eq. (3.17) in comparison with (1.55) to polar moment of inertia I_p (sometimes called a torsional constant). It determines a torsion resistance and it is essential to know its value. The value can be obtained either numerically from the volume of the membrane (see Section 3.2) or analytically. Some analytical solutions are known for common cross-sections, e.g. from [3].

Assume the cross-sections with the dimensions from Figure 3.4. The quantity p is a parameter and will be set as 1. The analytic values of the polar moment of inertia for parameter p are written below.

Circular cross-section

$$I_{p_a} = \frac{1}{2}\pi p^4 \tag{3.18}$$

Square cross-section

$$I_{p_a} = 2.25 p^4 \tag{3.19}$$

Equilateral triangle cross-section

$$I_{p_a} = \frac{\sqrt{3}(2p)^4}{80} \tag{3.20}$$

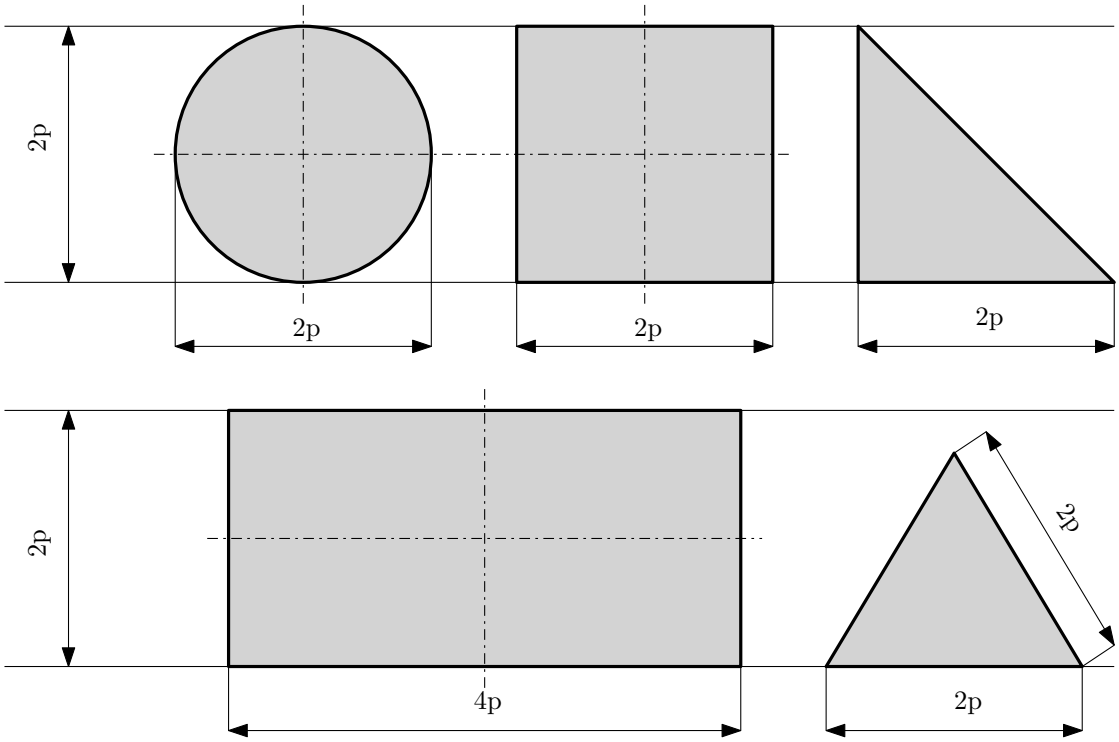


Figure 3.4: Parametric study

Rectangular cross-section

$$I_{p_a} = 2p \cdot p^3 \left[\frac{16}{3} - 3.36 \frac{p}{2p} \left(1 - \frac{p^4}{12(2p)^4} \right) \right] \tag{3.21}$$

$$I_{p_a} = 7.32416667 p^4 \tag{3.22}$$

Right-angled triangle cross-section

$$I_{p_a} = 0.0915(\sqrt{2}p)^4 \left(\frac{\sqrt{8}p}{\sqrt{2}p} - 0.8592 \right) \tag{3.23}$$

$$I_{p_a} = 0.4175328 p^4 \ . \tag{3.24}$$

The analytic value of the moment of inertia will be compared with the numerical value from FEM and BEM. The error can be calculated as

$$\text{error} = \frac{|I_{p_a} - I_{p_n}|}{I_{p_a}} \cdot 100[\%] \tag{3.25}$$

where I_{p_n} is equal to $4V_u$ from TorPy.

3.2 Application in author's software TorPy + author's mesh generator

The software TorPy is able to evaluate an arbitrary cross-section with triangular or quadrilateral mesh with FEM. The output comprises from stresses, the rate of twist, the angle of twist and the torsional rigidity. The software is also able to transform the triangular mesh to BE mesh and automatically detect boundaries and apply the boundary conditions. The TorPy uses the membrane analogy and the theory shown in Chapter 1.

3.2.1 Author's Mesh generator

An own mesh generator was developed for testing the code for general cross-section.
The meshes were created for circular, rectangular (square), equilateral triangle and right-angled triangle cross-sections. It is possible to set up arbitrary dimensions and change the number of elements.

3.2.1a Rectangle

The numbering principle of the rectangular mesh is shown in Figures 3.5 and 3.7. The mesh for this type of the cross-section is created for triangular and quadrilateral elements. The length of the side in x-direction is called L_x and the length of the side in y-axis described as L_y. The number of elements refers to x-axis represents n_x and analogously n_y in y-direction. Obviously, h_x is the length of one element in x-direction and analogously, h_y in y-axis. The triangular mesh is created by three nodes per element. For instance, the element 14 consists of nodes 8, 9 and 16.

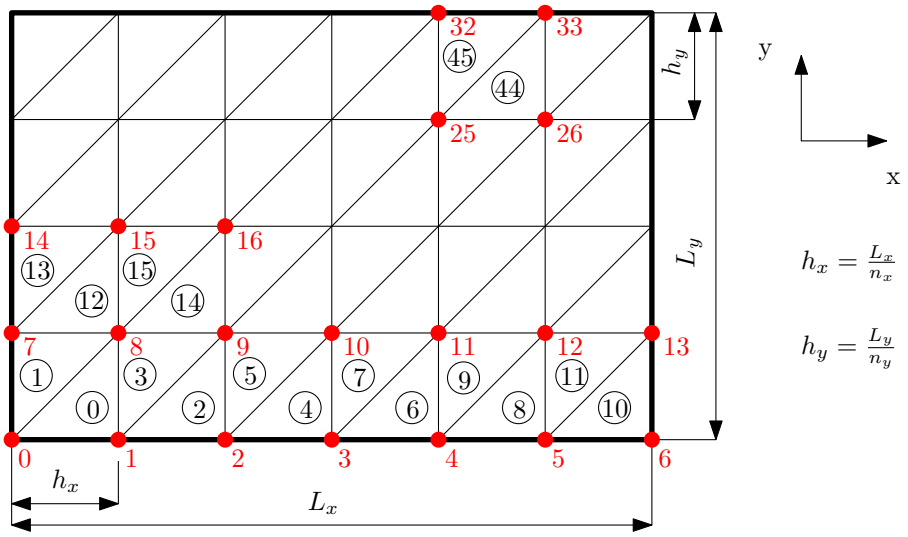


Figure 3.5: Rectangle - finite element mesh

Figure 3.6 illustrates the output from TorPy. It is clear, that L_x measures 140 mm, n_x is

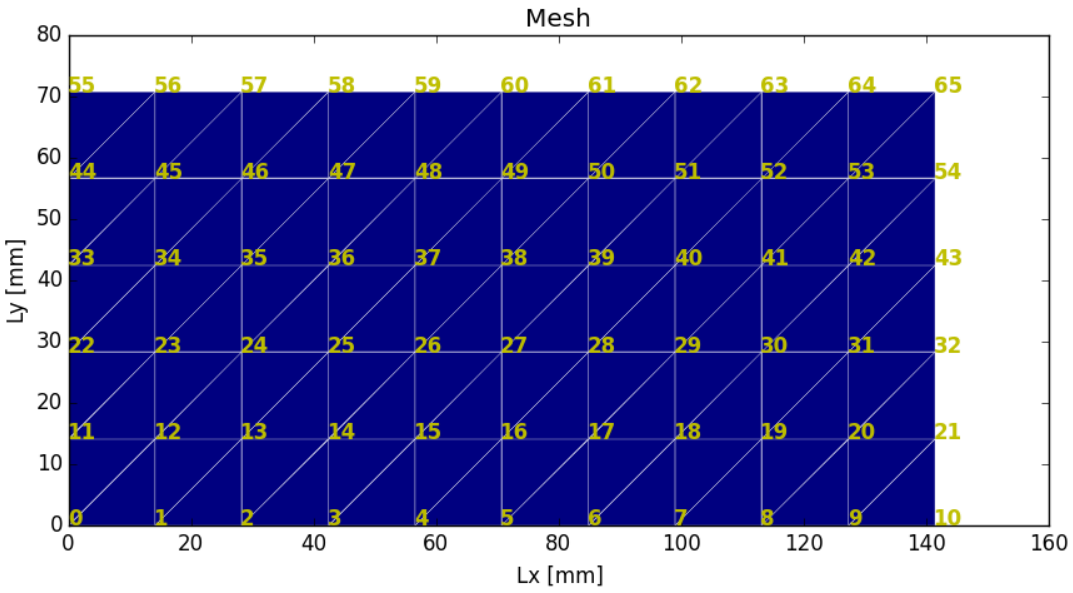


Figure 3.6: Mesh of rectangle cross section

equal to 10, L_y is 70 mm and n_y corresponds to 5. Likewise, the mesh for quadrilateral elements is created. One element contains four nodes. For example, the element 22 is assembled by nodes 25, 26, 33 and 32.

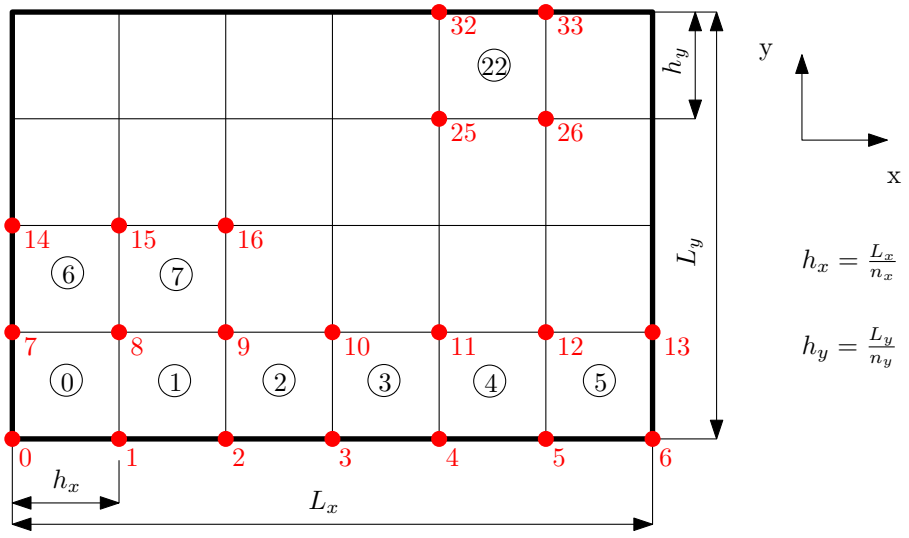


Figure 3.7: Rectangle - finite element mesh

3.2.1b Right-angled triangle

The inputs for right-angled triangular mesh consists of a length a and the number of rows. The length a represents the leg of triangle. The principle of numbering is shown in Fig. 3.8.

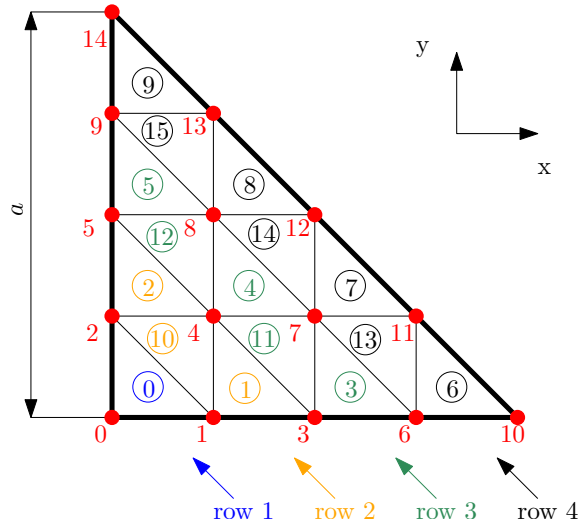


Figure 3.8: Right-angled triangle - finite element mesh

The output from TorPy shows that the length a is equal to 140 mm and the number of rows is 6 (see Fig. 3.9).

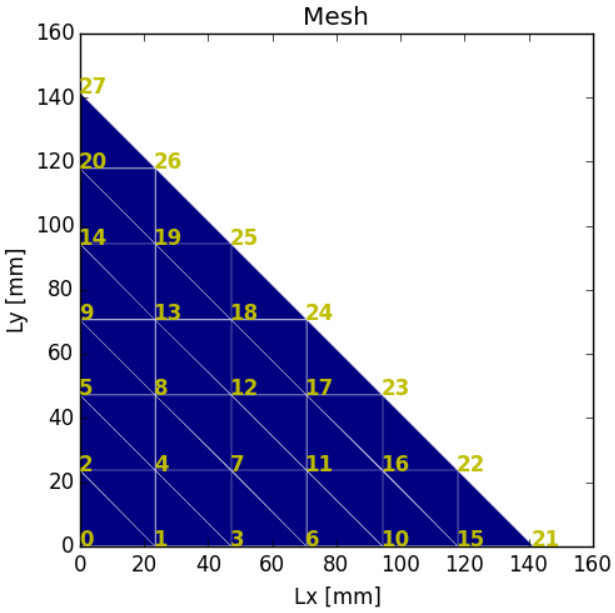


Figure 3.9: Mesh of right-angled triangle cross section

3.2.1c Equilateral triangle

The mesh of equilateral triangle is a modification of the previous mesh. The numbering of nodes and elements is the same as for right-angled triangle. Only coordinates are changed. The mesh of equilateral triangle is shown in Fig. 3.10.

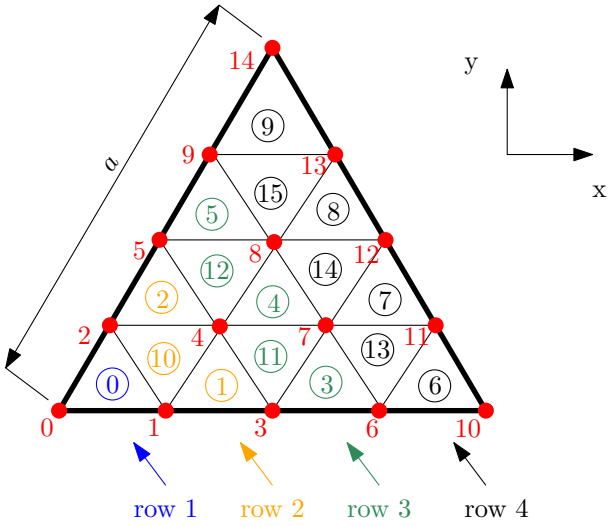


Figure 3.10: Right-angled triangle - finite element mesh

The Fig. 3.11 from software TorPy shows, that the length a is equal to 150 mm and cross-section contains 8 rows.

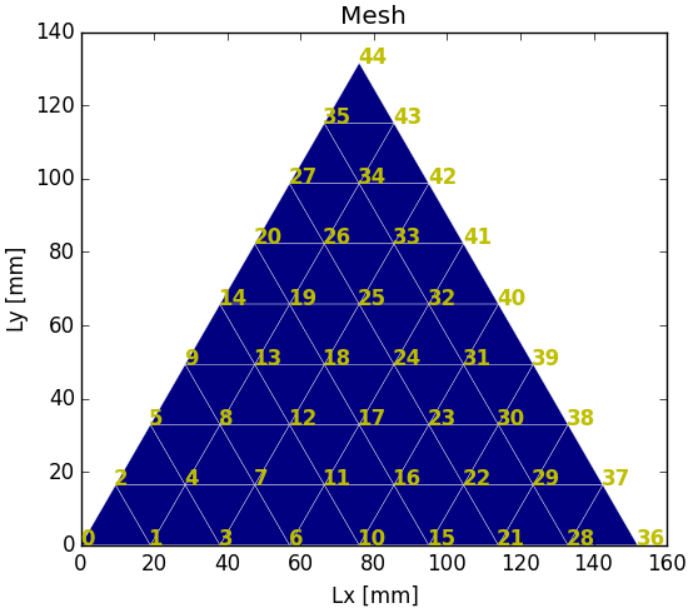


Figure 3.11: Mesh of equilateral triangle cross section

3.2.1d Circle

The rule of numbering is identical with triangular meshes, as shown in Fig. 3.12. Coordinates are transformed again and they are located in the circular curves (see red lines in Fig. 3.12).

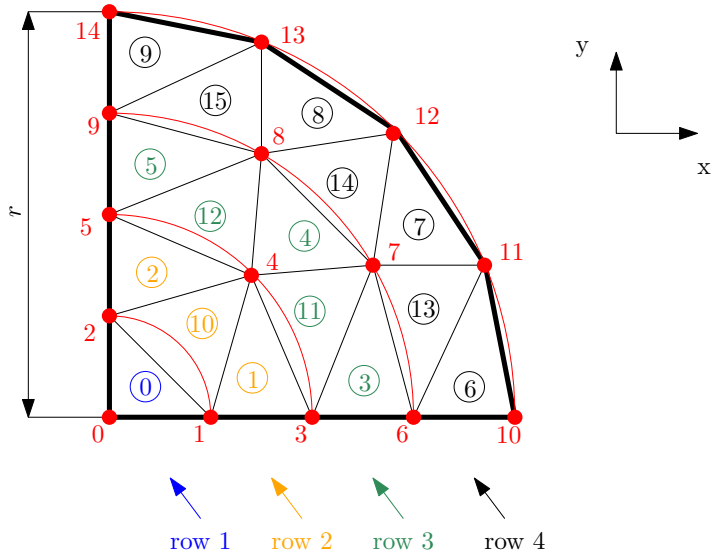


Figure 3.12: Circle - finite element mesh

The TorPy output is shown in Fig. 3.13, where the radius r is $r = 55$ mm and the mesh includes 10 rows.

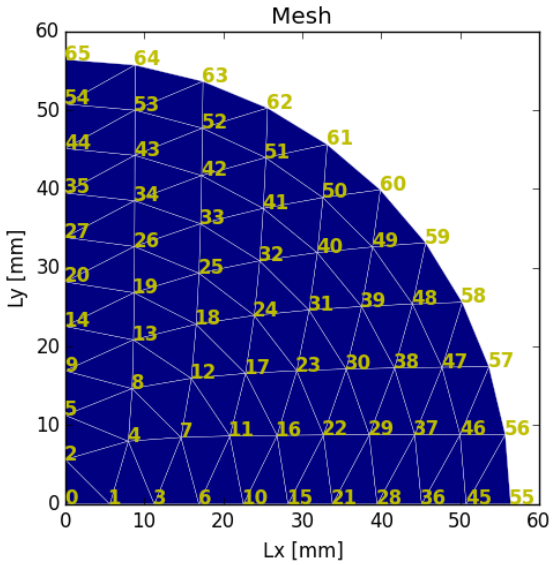


Figure 3.13: Mesh of circle cross section (one quarter)

3.2.2 Comparison of the cross-section properties

The objective of this study is to evaluate the cross-section with the best property to torsion. The cross-sections shown in Fig. 3.4 will be compared and each bar must have an identical volume. Therefore, the area A is given and from this area can be calculated the dimensions of each cross-section. The dimensions for the general cross-section are given in Table 3.2

Name	Symbol	Size	Unit
Length	L	1	m
Area	A	0.01	m^2
Torque	T	10 000	Nm
Young's modulus	E	2.1e11	Pa
Poisson ratio	μ	0.3	-

Table 3.2: The task

The dimensions of the cross-sections are computed from the given area $A = 0.01m^2$.

3.2.2a Circle

$$A_{ci} = \pi r^2 \tag{3.26}$$

$$r = \sqrt{\frac{A_{ci}}{\pi}} \tag{3.27}$$

$$r = \sqrt{\frac{0.01}{\pi}} = 0.056419\,\text{m} \tag{3.28}$$

3.2.2b Square

$$A_{sq} = a^2 \tag{3.29}$$

$$a = \sqrt{A_{sq}} \tag{3.30}$$

$$a = \sqrt{0.01} = 0.1\,\text{m} \tag{3.31}$$

3.2.2c Rectangle

$$A_{rec} = ab = a \cdot 2a \tag{3.32}$$

$$a = \sqrt{\frac{A_{rec}}{2}} \tag{3.33}$$

$$a = \sqrt{\frac{0.01}{2}} = 0.07071\,\text{m} \tag{3.34}$$

3.2.2d Right-angled triangle

$$A_{trra} = \frac{1}{2}a^2 \tag{3.35}$$

$$a = \sqrt{2 \cdot A_{trra}} \tag{3.36}$$

$$a = \sqrt{2 \cdot 0.01} = 0.141421 \text{ m} \tag{3.37}$$

3.2.2e Equilateral triangle

$$A_{treq} = \frac{\sqrt{3}}{4}a^2 \tag{3.38}$$

$$a = 2\frac{\sqrt{A_{treq}}}{\sqrt[4]{3}} \tag{3.39}$$

$$a = 2\frac{\sqrt{0.01}}{\sqrt[4]{3}} = 0.151967 \text{ m} \ . \tag{3.40}$$

3.2.3 Membrane analogy- numerical polar moment of inertia

The numerical value of the polar moment of inertia will be compared with analytical value as written before in Section 3.1.5. The volume is calculated as the area of the element multiplied by an average displacement, i.e.

$$V^e = \phi_V^e \cdot A^e \tag{3.41}$$

where ϕ_V^e denotes the average displacement. It can be calculated for the rectangular element as

$$\phi_V^e = \frac{1}{4}\left(\phi_0 + \phi_1 + \phi_3 + \phi_4\right) \tag{3.42}$$

or for the triangular element

$$\phi_V^e = \frac{1}{3}\left(\phi_1 + \phi_5 + \phi_4\right) \ . \tag{3.43}$$

The numbering of the rectangular and triangular elements and the illustration of the volumes are shown in Fig. 3.14.

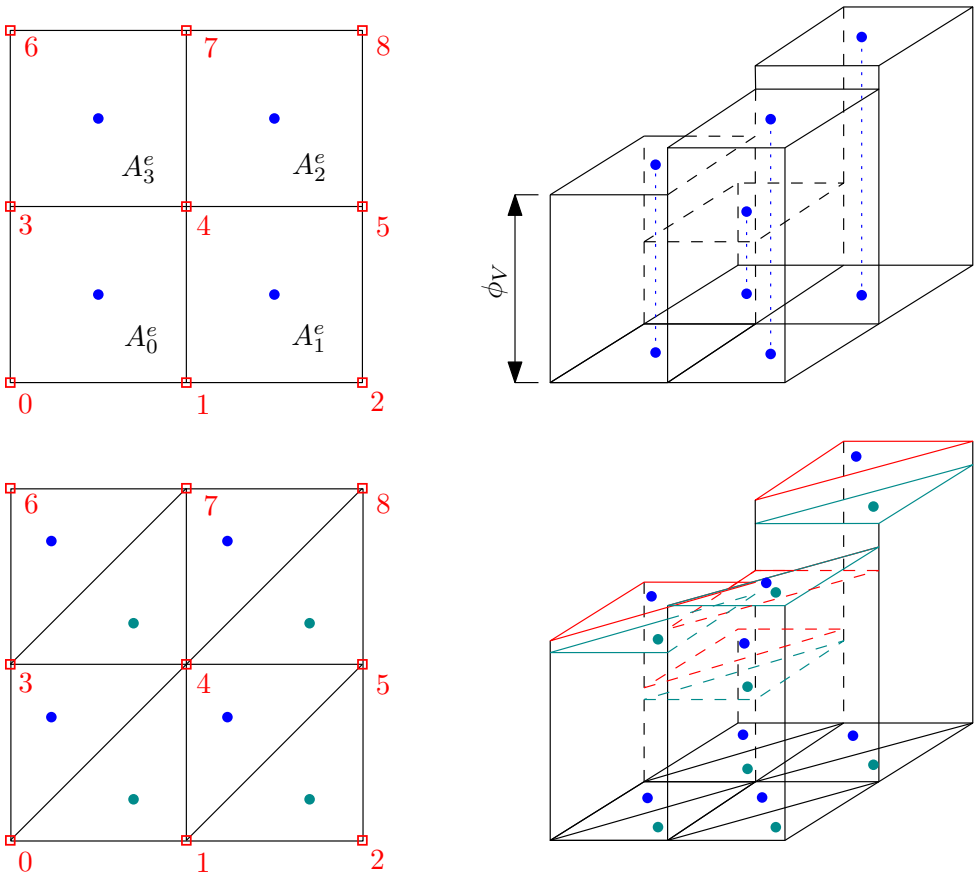


Figure 3.14: Volume calculation

3.3 Application in author's software TorPy + external mesh generator

A function was developed, which can transform a triangular mesh from MSC Patran or MSC Marc to TorPy friendly form. The code will be tested on a real problem and compared with commercial software, as shown below.

3.3.1 Comparison with commercial software Ansys

Suppose a shaft with a groove for a key (keyseat), as shown in Fig. 3.15. A consequence of the keyseat is a notch, which causes the stress concentration. The stress maximum is expected in the vicinity of radius R_1. In this location must be fine mesh.

The dimensions t, b, R_1 are taken from [21].

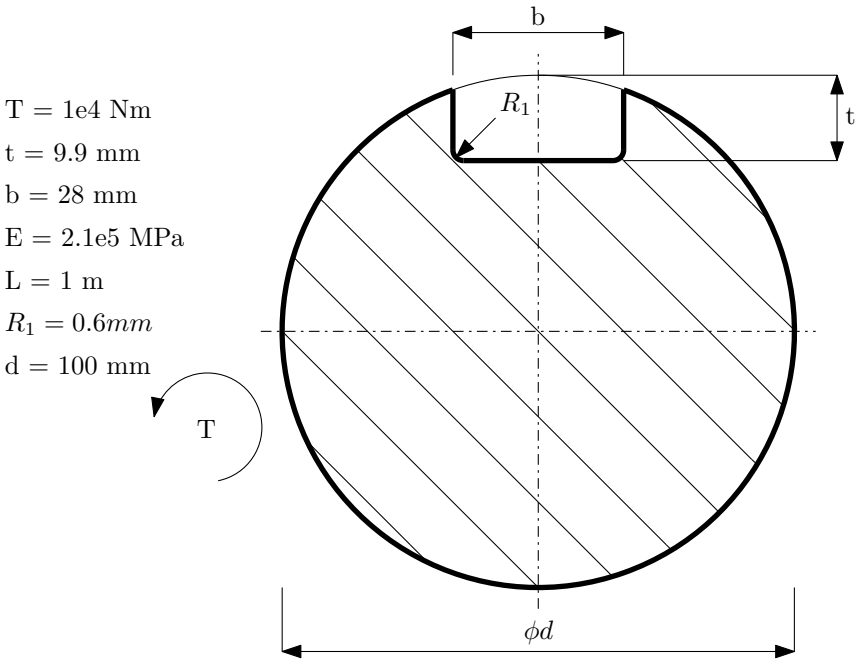


Figure 3.15: Key groove shaft

3.3.2 Modelling of the torsion in software Ansys

In commercial software, three-dimensional problem has to be modelled. The torque is applied at one end of the beam and the other end is fixed (see Fig. 3.16). It means, the reaction causes a torque in opposite direction.

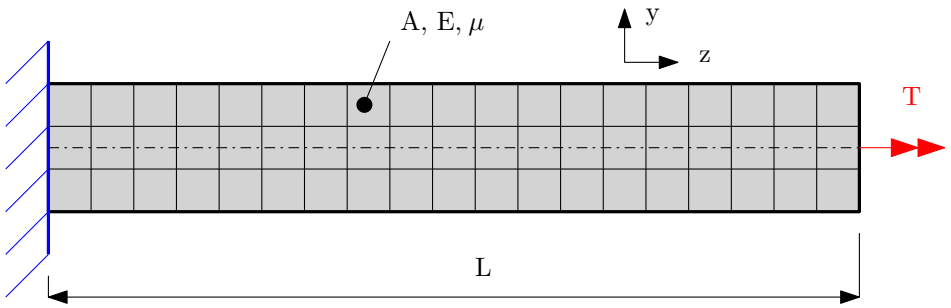


Figure 3.16: Boundary conditions

The evaluation of the stress is carried out in the middle of the length of beam due to Saint-Venant principle, as shown in Figure 3.17.

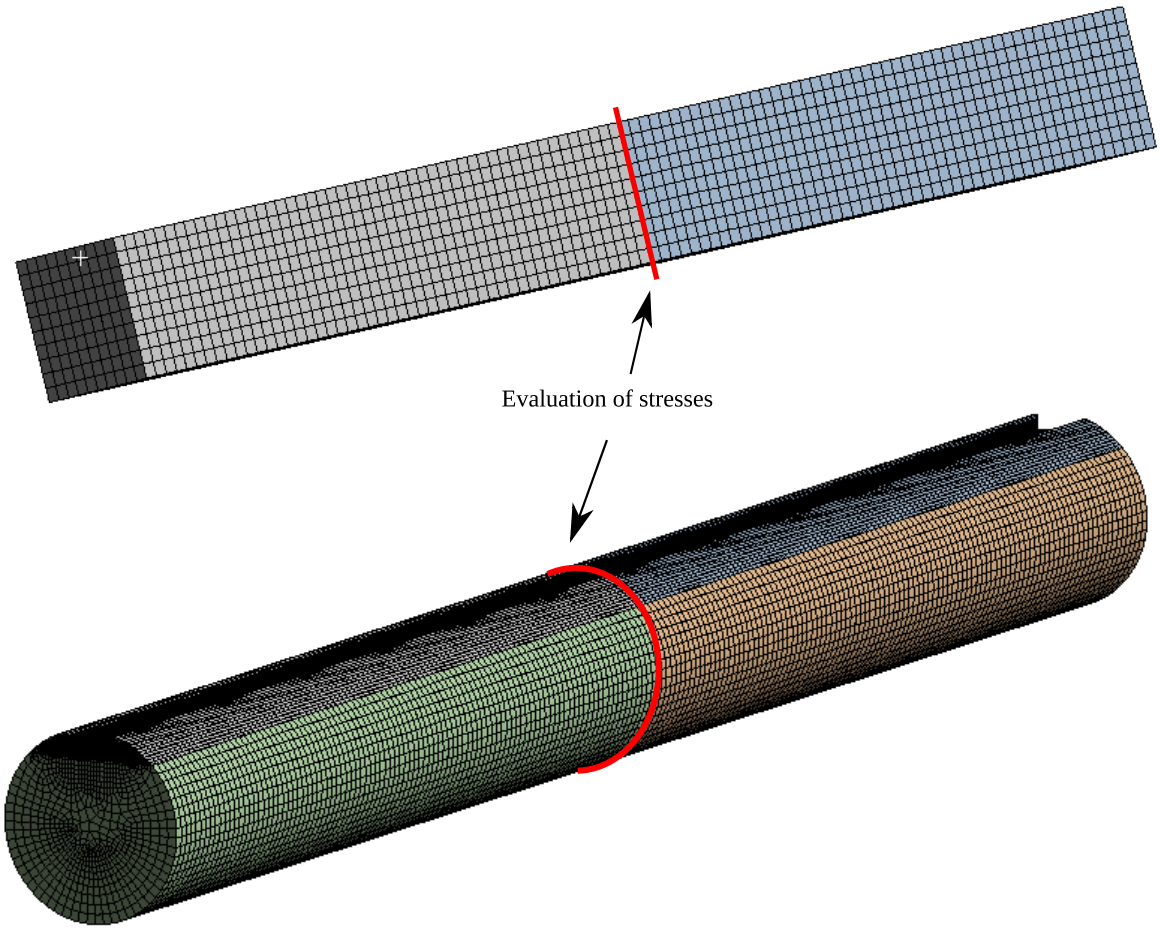


Figure 3.17: Mesh

The analysis of the shaft with keyseat contained in Ansys 347 800 elements (Solid186) and 1 427 520 nodes. The TorPy mesh had 5298 elements and 2863 nodes. It might be applied symmetry on the cross-section in Fig. 3.15, however, the author used a function to detect the boundary automatically. The function also applies the boundary conditions.

The mesh of the cross-sections are shown in Fig. 3.18. The mesh is refined in the corners, since there is the assumption of the maximum stress.

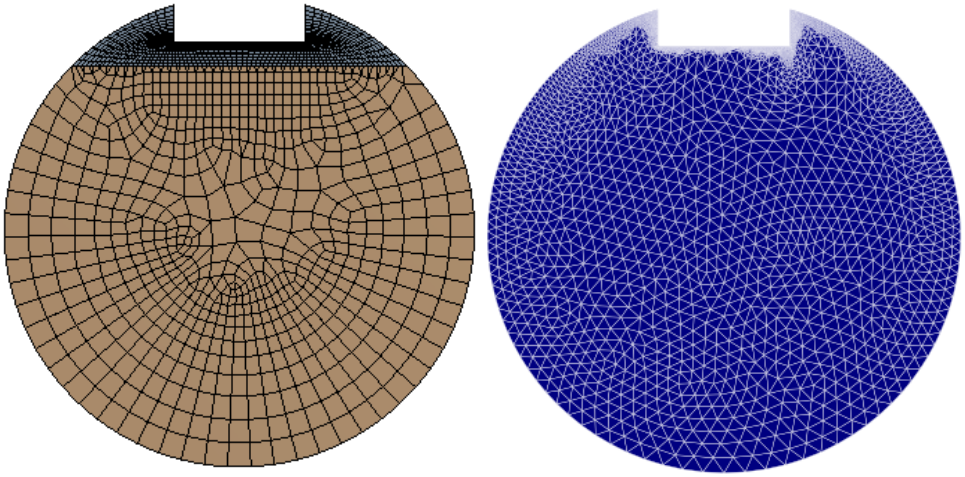


Figure 3.18: Comparison of meshes from Ansys and TorPy

Chapter 4

Results

The results of problems from Chapter 3 are introduced in this chapter.

4.1 Application in author's software TorPy + own mesh generator

In previous chapter, there was introduced the own mesh generator in Section 3.2.1 and the results will be presented here.

4.1.1 Comparison of the cross-section properties

Figure 4.1 illustrates the results of maximum stress of the cross-sections with same volume (see properties in Table 3.2). The most suitable cross-section is the circular cross-section, as expected. The torsion of circular cross-section does not cause the warping and the resulting stress is the smallest. Note the first stress maximum of circular cross-section in Fig. 4.1. The value is larger than the result with finer mesh and it is caused inaccuracy of the mesh (cf. 3.12 and 3.13). The total area is less and consequently, the stress is larger.

It exists a lot of methods to evaluate stresses in quadrilateral element. It was chosen simple method, where the stress is calculated in Gaussian points and averaged within element. It means that the resulting stress is constant over elements.

The convergence is shown in Figure 4.1. The more elements per edge it is, the solution is the more accurate, i.e. it converges to exact solution. The direction angle of the lines approaches to zero.

The stress distribution of each cross-section and the analytical solution of stress τ is shown below. The stresses from Ansys are evaluated in the middle of beam, as shown in Fig. 3.17.

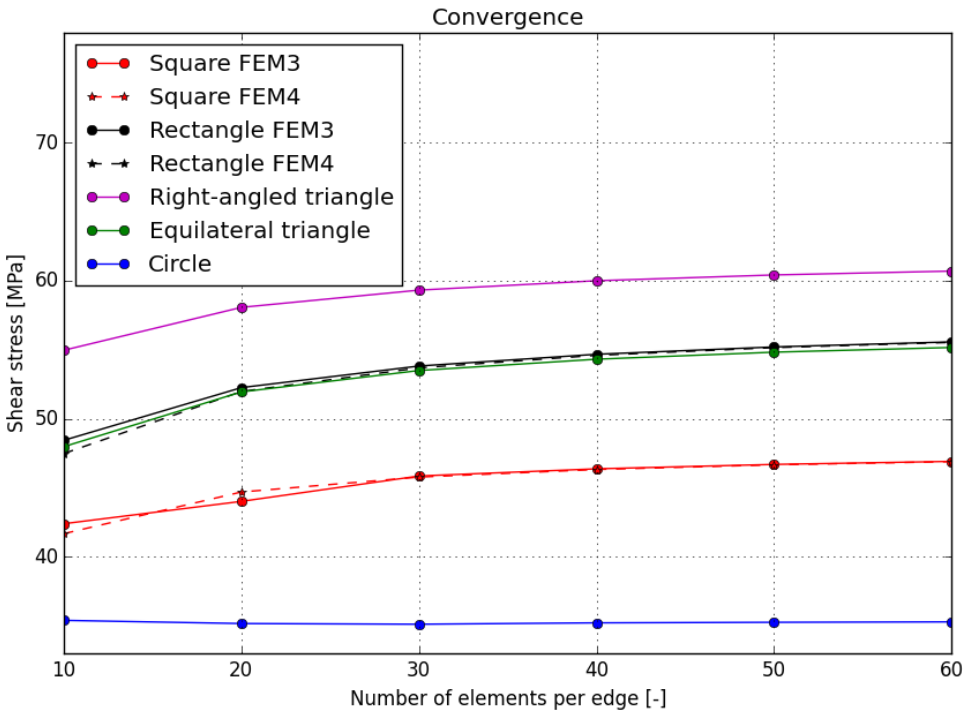


Figure 4.1: Convergence

4.1.1a Circular cross-section

The analytical solution of the maximum shear stress is according to Eq. (3.1) 35.45 MPa. The figures on the left side are from software Ansys and the figures on the right side are from TorPy.

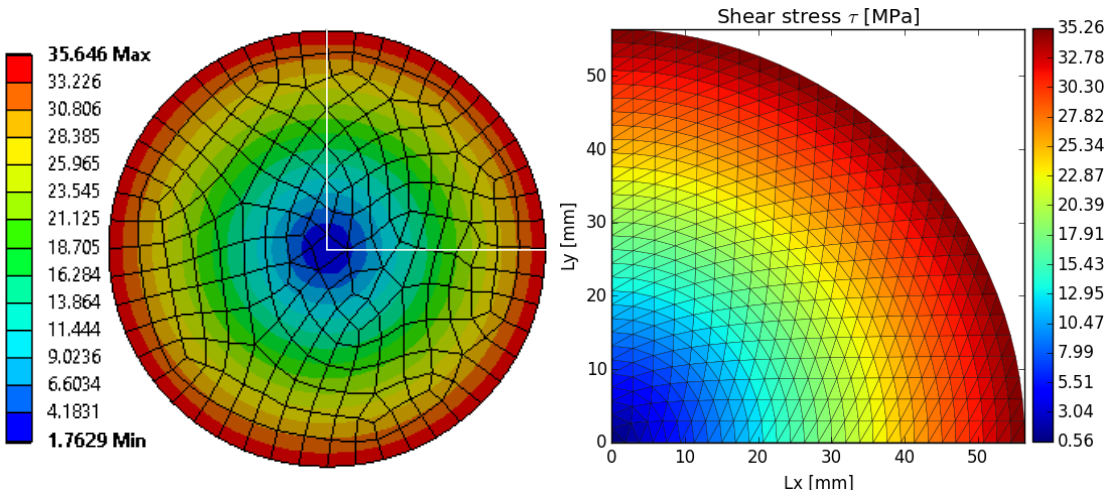


Figure 4.2: Maximum shear stress τ_{max} of the circle (Ansys - TorPy)

The mesh from Ansys is coarse, however, quadratic elements are employed. Due to suitable boundary conditions, only one-quarter of the cross-section is analyzed in TorPy. The results are accurate. The shear stress τ_{xz} varies from -35 MPa to 35 MPa, as shown in Fig. 4.3.

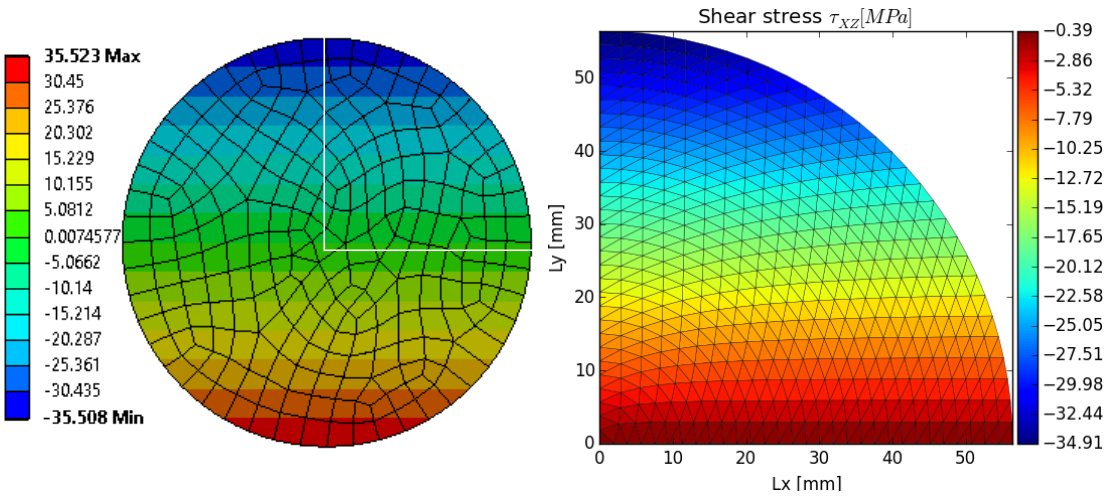


Figure 4.3: Shear stress τ_{xz} of the circle (Ansys - TorPy)

The shear stress τ_{yz} is similar to stress distribution τ_{xz} and rotated on 90 degrees (see Fig. 4.4).

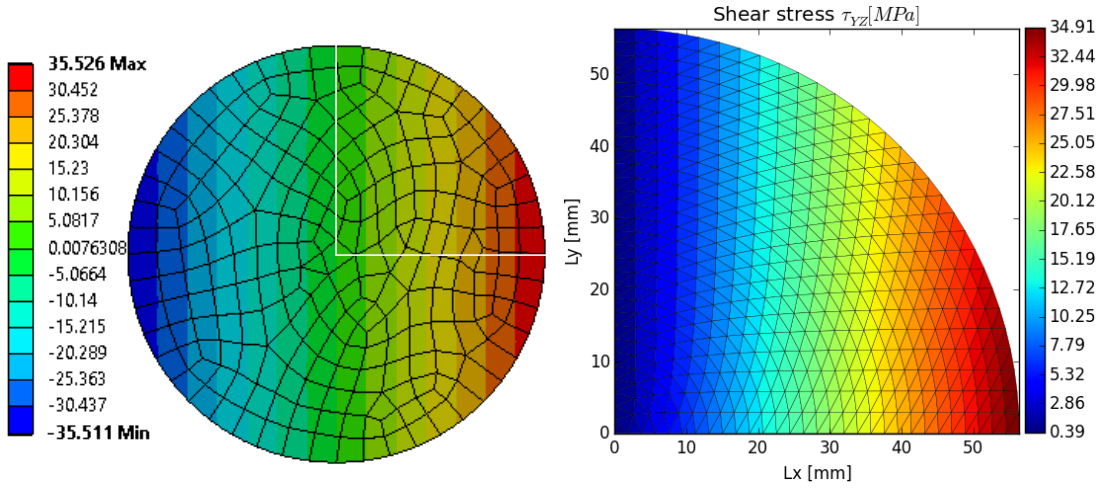


Figure 4.4: Shear stress τ_{yz} of the circle (Ansys - TorPy)

4.1.1b Square cross-section

The analytical solution of τ_{max} is in accordance with Eq. (3.3) equal to 48.08 MPa. Ansys used 10 quadratic elements SOLID186 per side and TorPy used 60 elements per edge. However, a change of the maximum stress of 30 elements per edge and 60 elements per edge is according to Fig. 4.1 negligible.

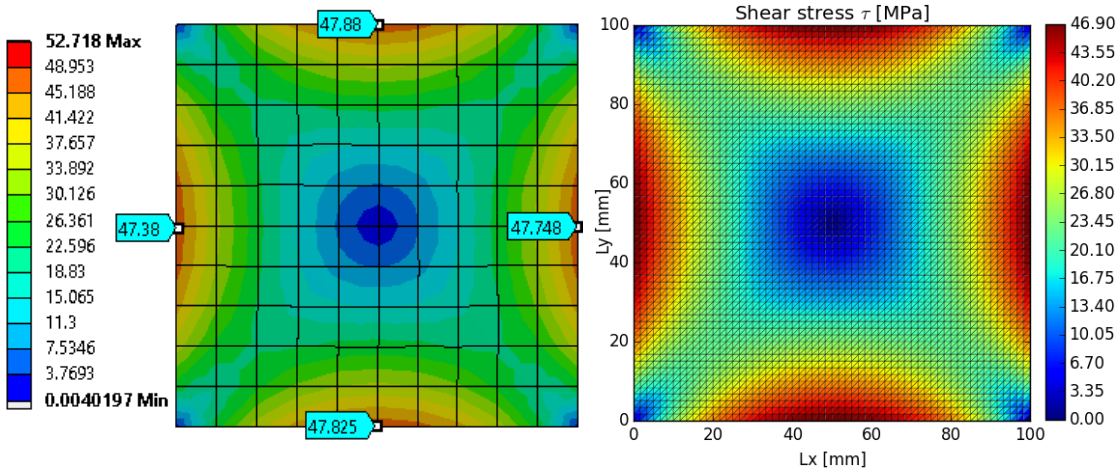


Figure 4.5: Maximum shear stress τ_{max} of the square (Ansys - TorPy)

The range of legend in Ansys is influenced by boundary conditions, therefore, the maximum stress is labeled on the cross-section. The difference between TorPy solution and analytical solution is less than 3%.

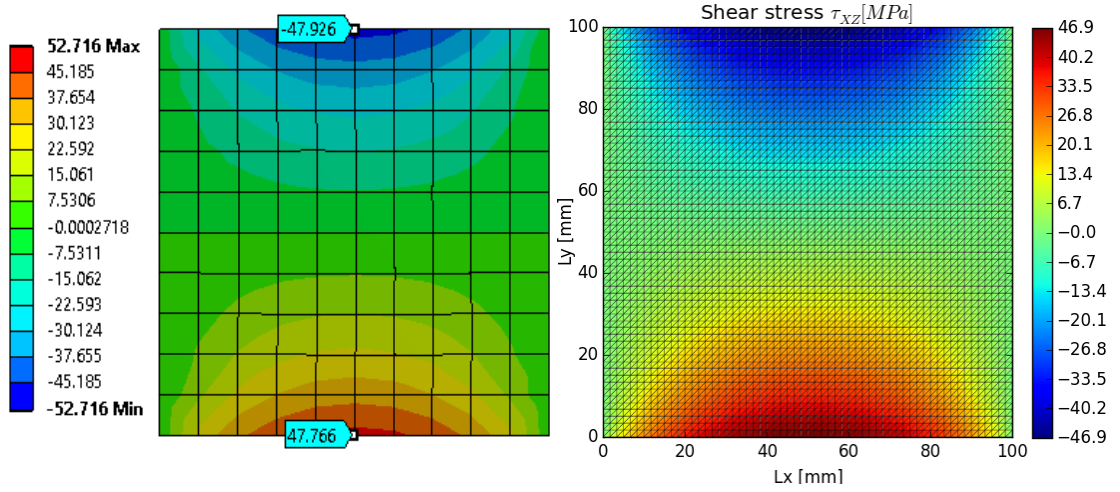


Figure 4.6: Shear stress τ_{xz} of the square (Ansys - TorPy)

The stress distributions from Ansys and TorPy in Figs. 4.5 through 4.7 are identical.

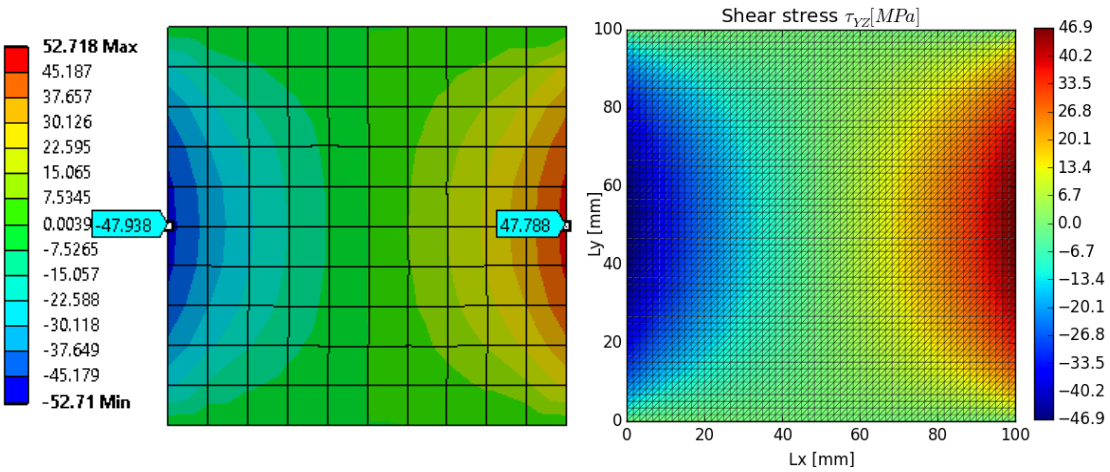


Figure 4.7: Shear stress τ_{yz} of the square (Ansys - TorPy)

4.1.1c Rectangle cross-section

The analytical solution of τ_{max} is on the longer side is equal to 57.49 MPa and on the shorter side is τ equal to 45.59 MPa (see Eq. (3.3)).

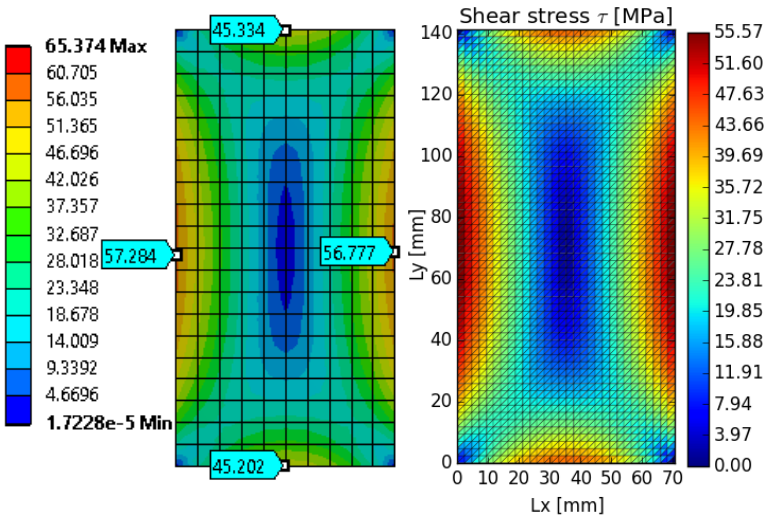


Figure 4.8: Maximum shear stress τ_{max} of the equilateral rectangle (Ansys - TorPy)

The shorter side is divided into 9 quadratic elements and 18 elements are used on the longer side of rectangle. The TorPy rectangle is divided into 60 elements on longer side and the shorter side is half the size as longer.

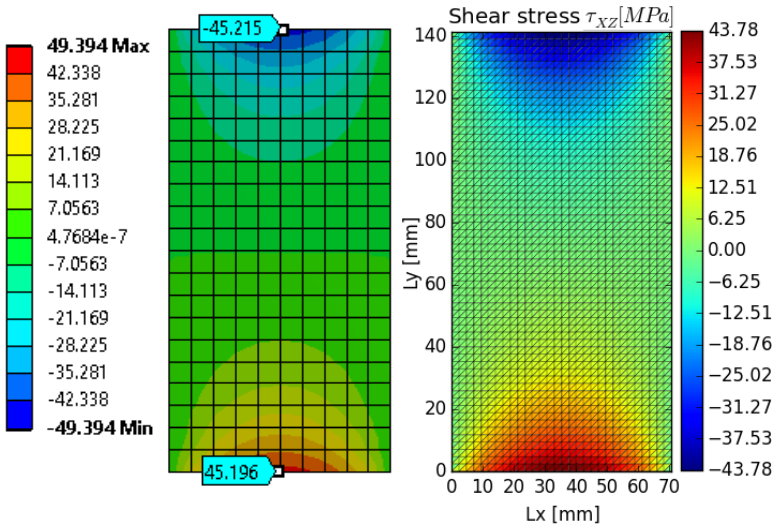


Figure 4.9: Shear stress τ_{xz} of the rectangle (Ansys - TorPy)

The difference of maximum stress between Ansys and TorPy is 3.3 %. The stress distributions of Ansys corresponds the stress distributions of TorPy as shown in Figures 4.8 through 4.10.

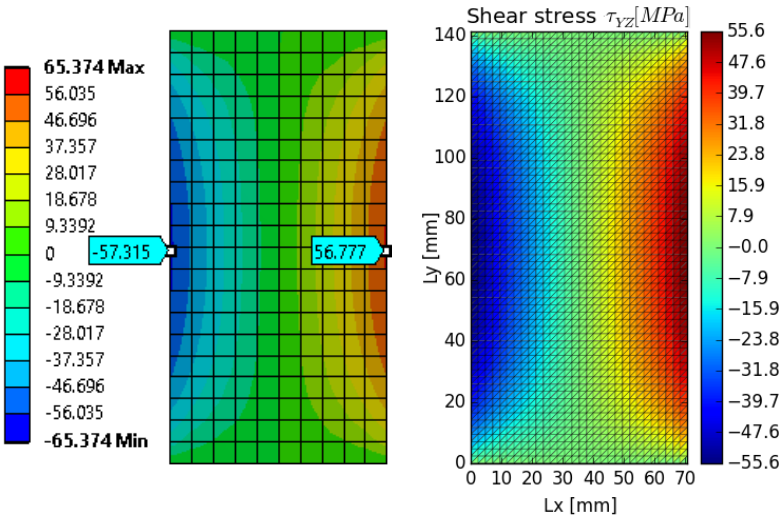


Figure 4.10: Shear stress τ_{yz} of the rectangle (Ansys - TorPy)

The stress maximums of quadrilateral elements are shown in Fig. 4.11. The visualization is executed in software Paraview, since it is simpler than in Python. The number of elements per edge is 50. Differences in results between analytical and quadrilateral elements are negligible.

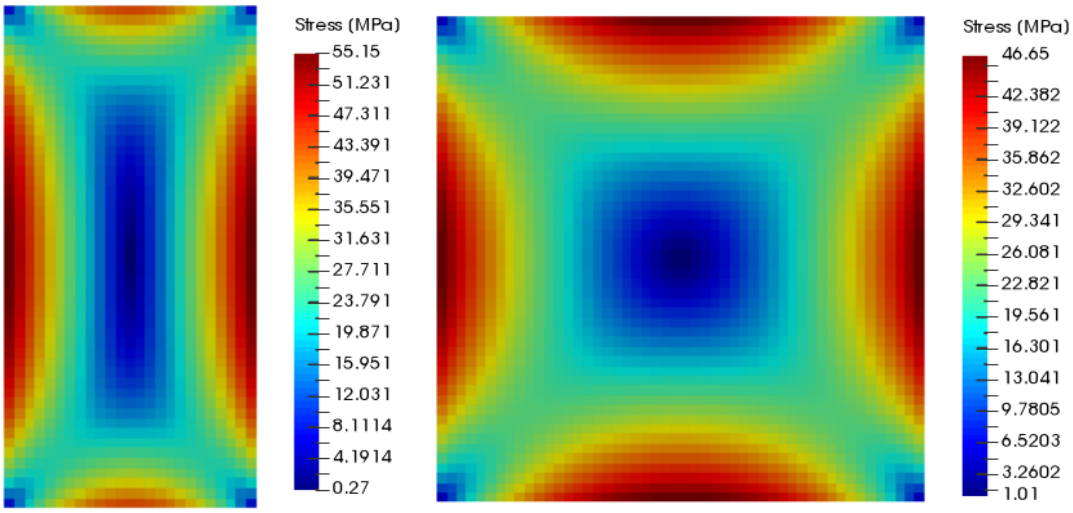


Figure 4.11: Maximum stress τ_{max} of the rectangle and square with quadrilateral elements

4.1.1d Equilateral triangle cross-section

According to Eq. (3.5), the maximum stress is equal to 56.99 MPa. The difference between analytical solution and TorPy solution is equal to 3.2%.

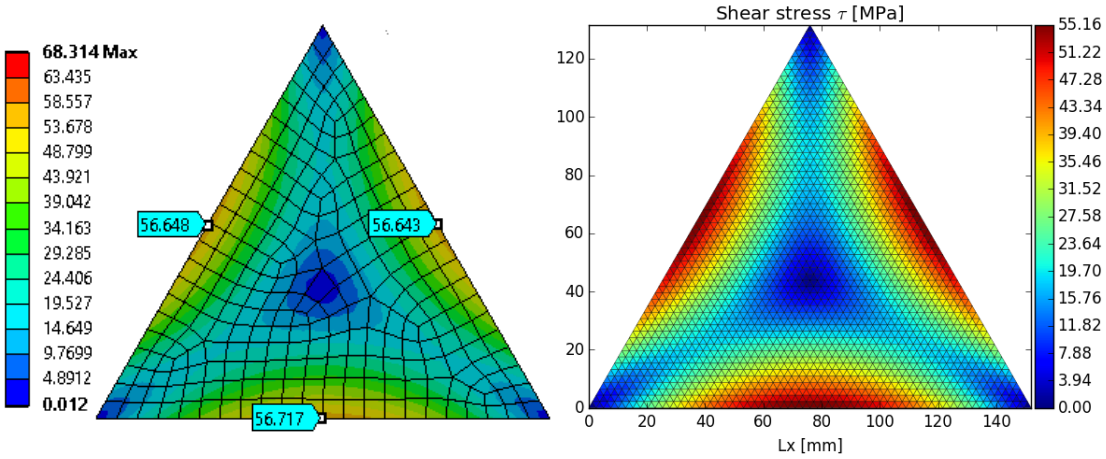


Figure 4.12: Maximum shear stress τ_{max} of the equilateral triangle (Ansys - TorPy)

The stress distributions of the stresses from Ansys and TorPy are shown in Figs. 4.12 through 4.14.

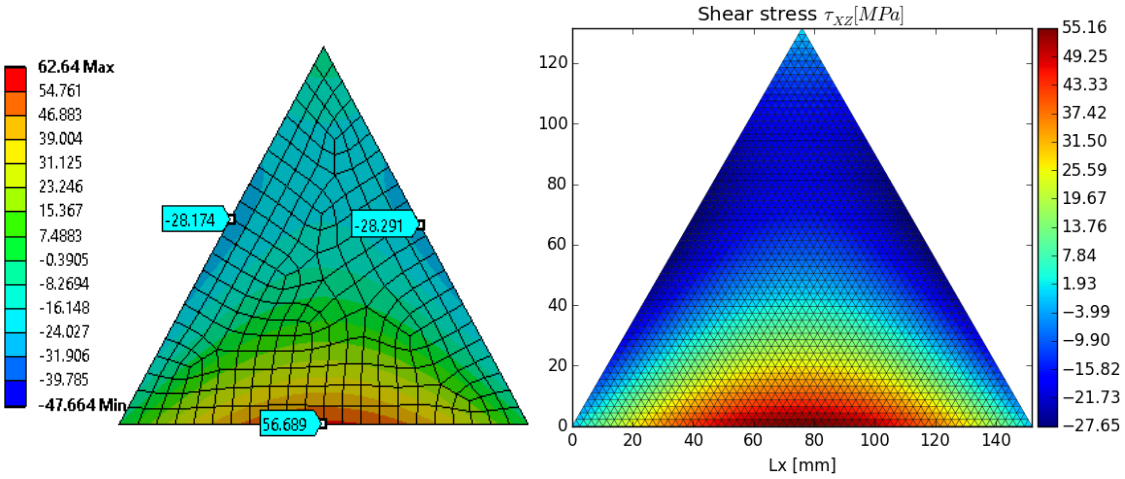


Figure 4.13: Shear stress τ_{xz} of the equilateral triangle (Ansys - TorPy)

Ansys used 21 quadratic elements per edge and TorPy divided edge to 60 linear elements.

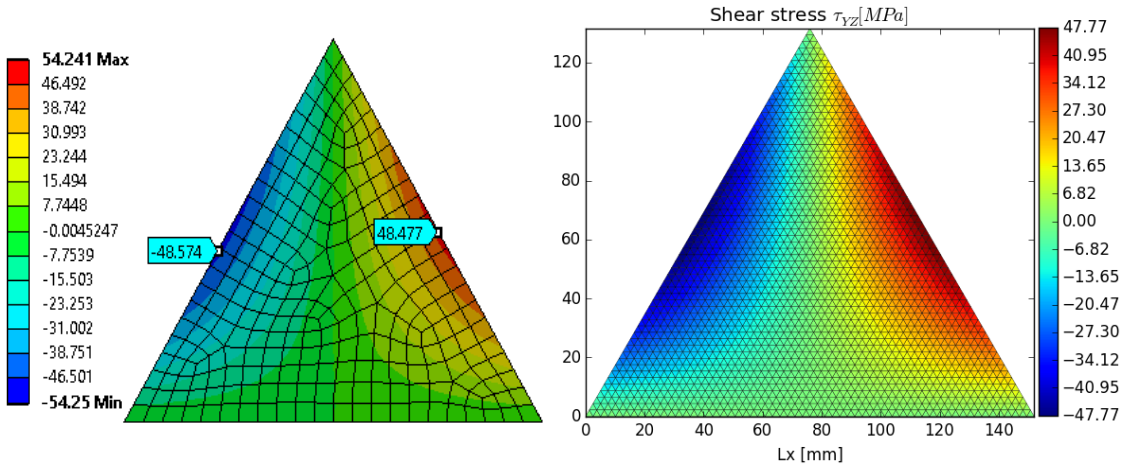


Figure 4.14: Shear stress τ_{yz} of the equilateral triangle (Ansys - TorPy)

4.1.1e Right-angled triangle cross-section

The maximum shear stress τ_{max} is according to Eq. 3.7 equal to 61.98 MPa. The difference between TorPy solution and exact solution is equal to 2%.

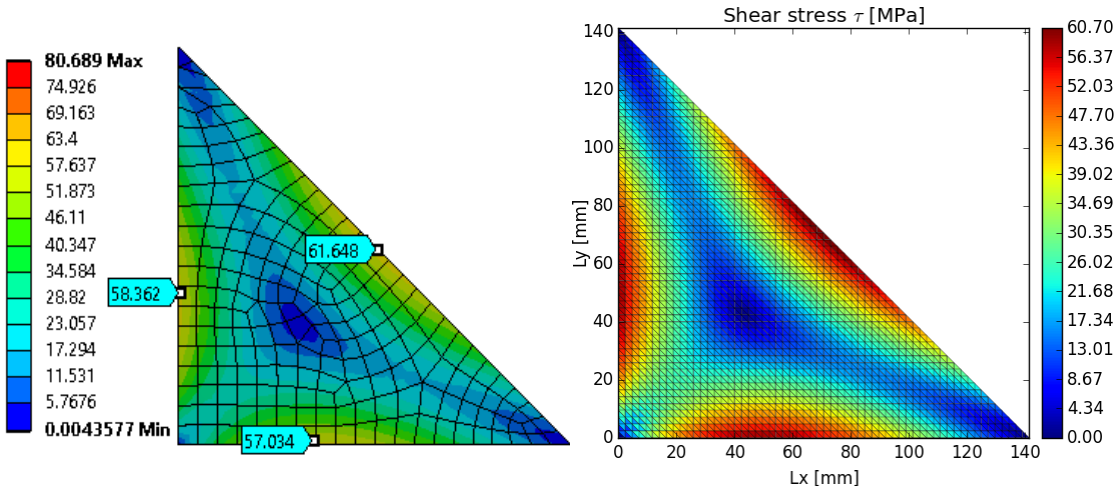


Figure 4.15: Maximum shear stress τ_{max} of the right-angled triangle (Ansys - TorPy)

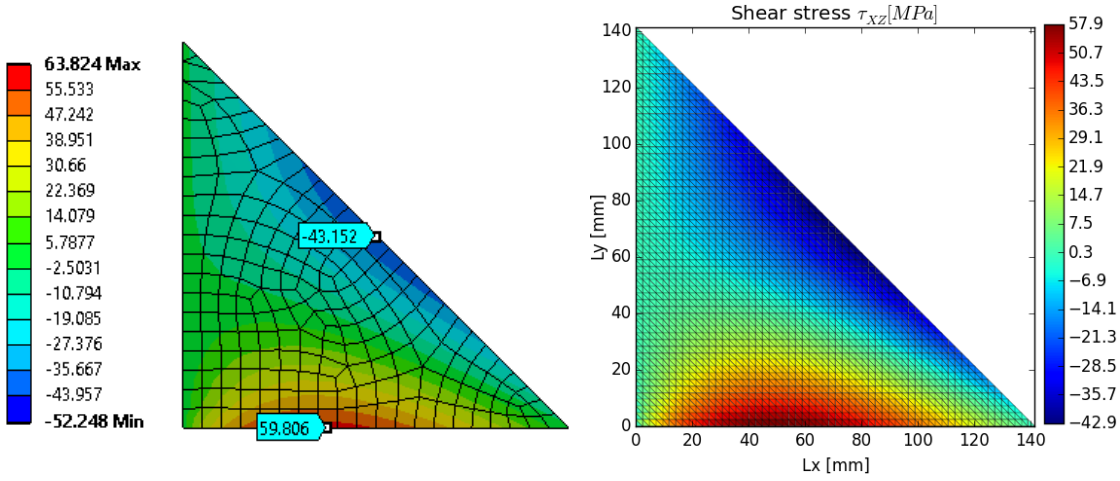


Figure 4.16: Shear stress τ_{xz} of the right-angled triangle (Ansys - TorPy)

The leg of triangle is divided into 20 quadratic elements and TorPy leg contains 60 linear elements. The stress distributions of τ_{max}, τ_{xz} and τ_{yz} are shown in Figs. 4.15 through 4.17.

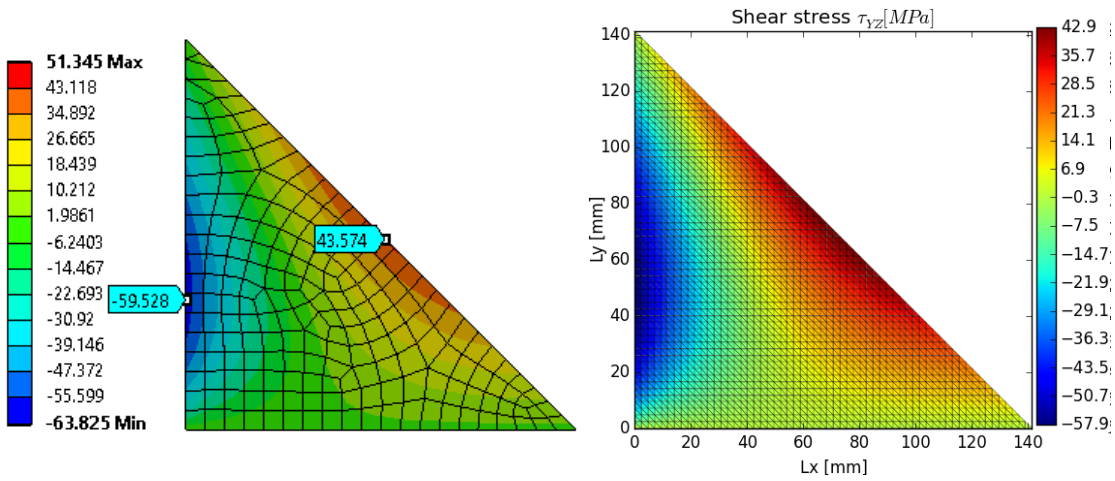


Figure 4.17: Shear stress τ_{yz} of the right-angled triangle (Ansys - TorPy)

The stress distribution along the entire length of the square cross-section is shown in Fig. 4.18. It is clear that both ends are effected by boundary conditions.

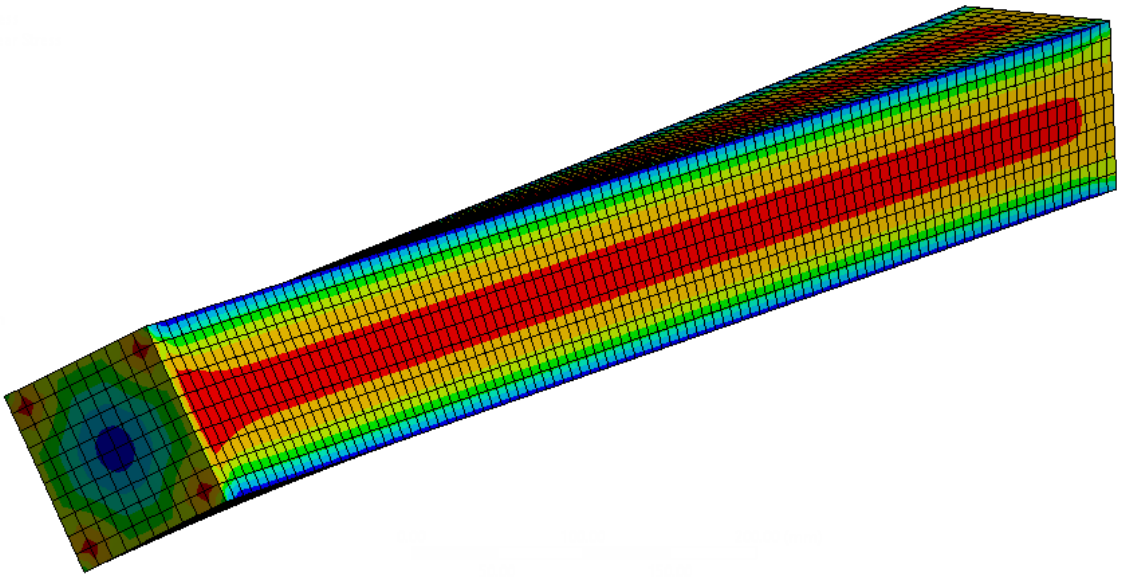


Figure 4.18: Distribution of stress along the length - Ansys

4.1.2 Membrane analogy - numerical polar moment of inertia

The membranes are shown for 30 elements per edge from triangular finite element method from TorPy. The membranes from BEM and FEM4 look identical, since they are tested on same finite element mesh.

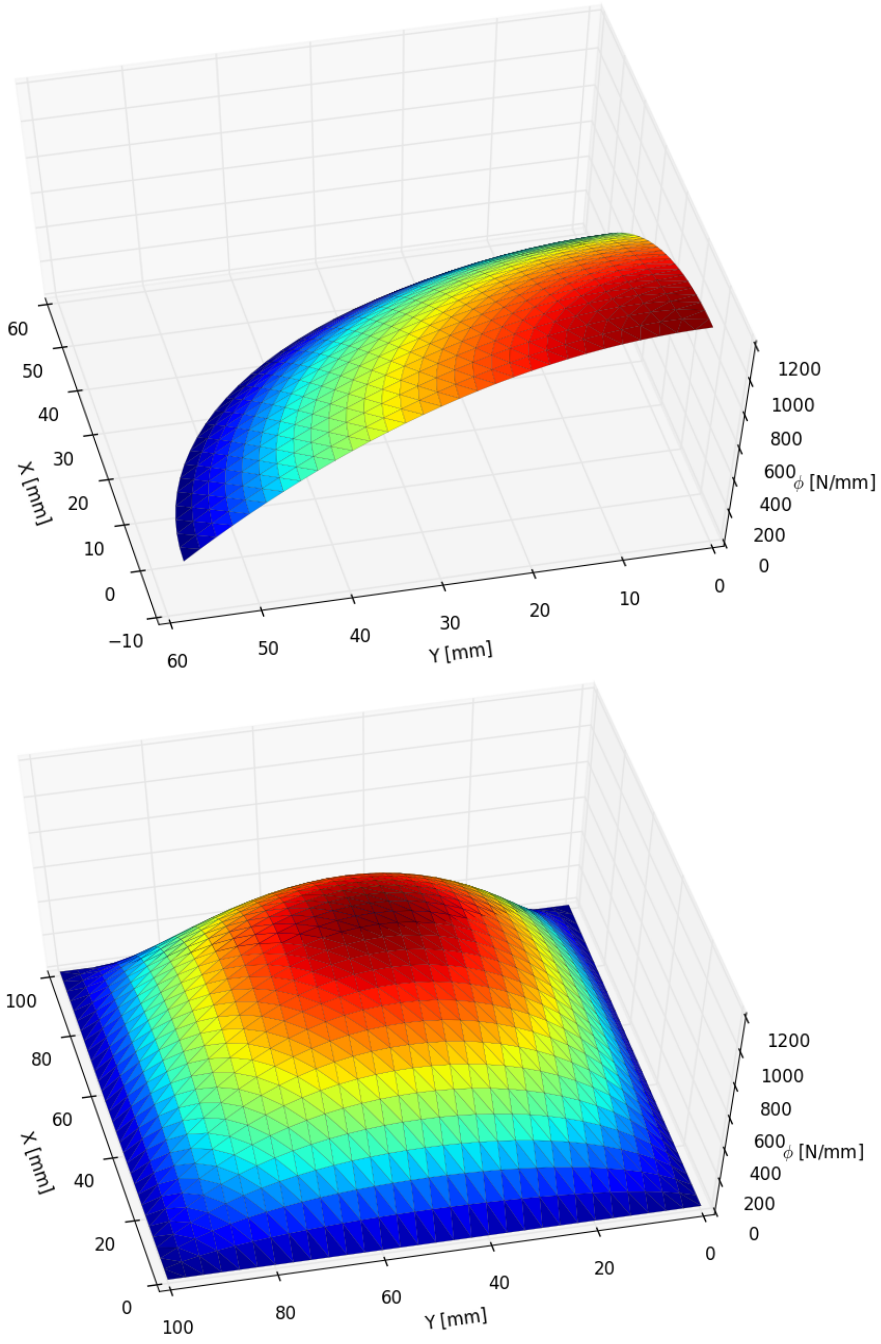


Figure 4.19: Membranes over one-quarter of circle and square cross-sections

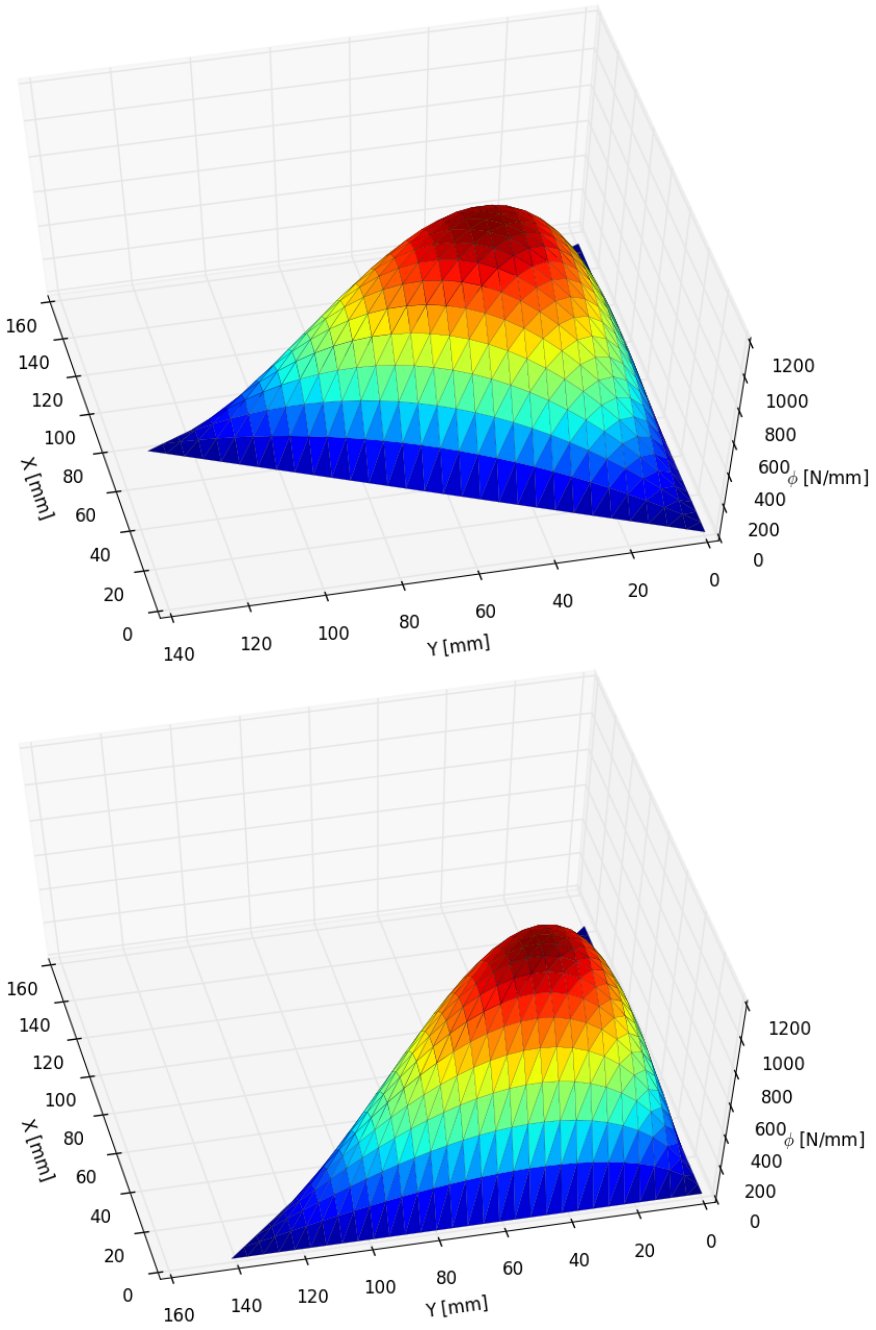


Figure 4.20: Membranes over equilateral triangle and right-angled triangle cross-sections

The difference between two methods is illustrated on square cross-section with 15 elements per edge. The error is calculated as

$$error = \frac{|\phi_{m1} - \phi_{m2|}}{\phi_{m1}} \cdot 100 \tag{4.1}$$

where $m1$ denotes method 1 (FEM3 and FEM4) and $m2$ represents method 2 (FEM4 and BEM) and the error is shown in Fig. 4.21.

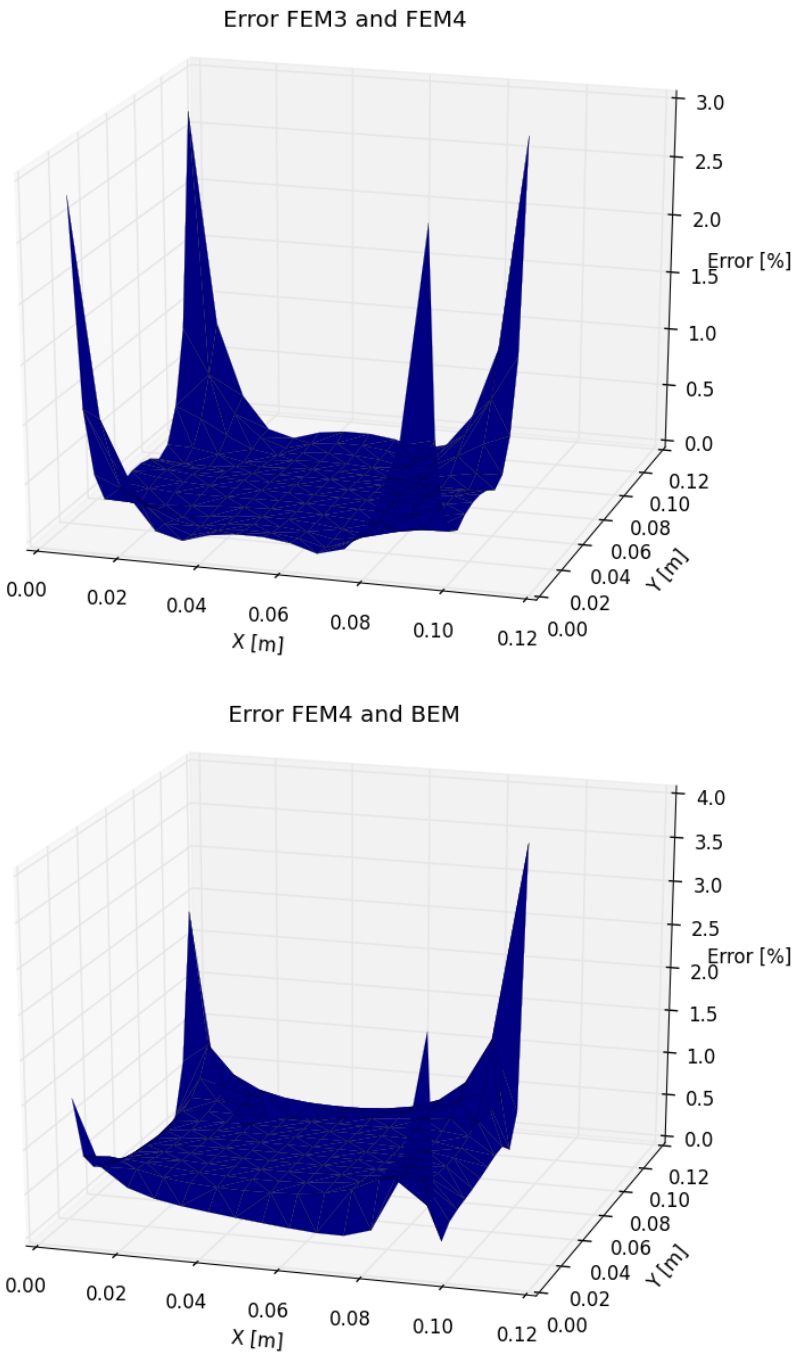


Figure 4.21: Error

CS	Method	Analytical	Numerical (n =)					
			10	20	30	40	50	60
Circle	FEM 3	1.5708	1.496	1.552	1.562	1.566	1.567	1.569
	BEM		1.452	1.541	1.558	1.564	1.567	1.568
Square	FEM 3	2.2500	2.178	2.231	2.241	2.245	2.246	2.247
	FEM 4		2.216	2.241	2.246	2.247	2.248	2.248
	BEM		2.202	2.238	2.244	2.247	2.248	2.248
Rectangle	FEM 3	7.3242	6.802	7.185	7.258	7.284	7.296	7.303
	FEM 4		7.006	7.240	7.283	7.298	7.305	7.309
	BEM		6.939	7.224	7.277	7.296	7.304	7.309
Right-angled tr.	FEM 3	0.4175	0.395	0.412	0.415	0.416	0.417	0.417
	BEM		0.391	0.408	0.412	0.414	0.415	0.415
Equilateral tr.	FEM 3	0.3464	0.329	0.342	0.344	0.345	0.346	0.346
	BEM		0.330	0.342	0.344	0.345	0.345	0.346

Table 4.1: Analytical and numerical value of the polar moment of inertia

The values of polar moment of inertia are arranged in Tab. 4.1. It is clear, that numerical values converges to analytical with more elements per edge, as shown in Fig. 4.22.

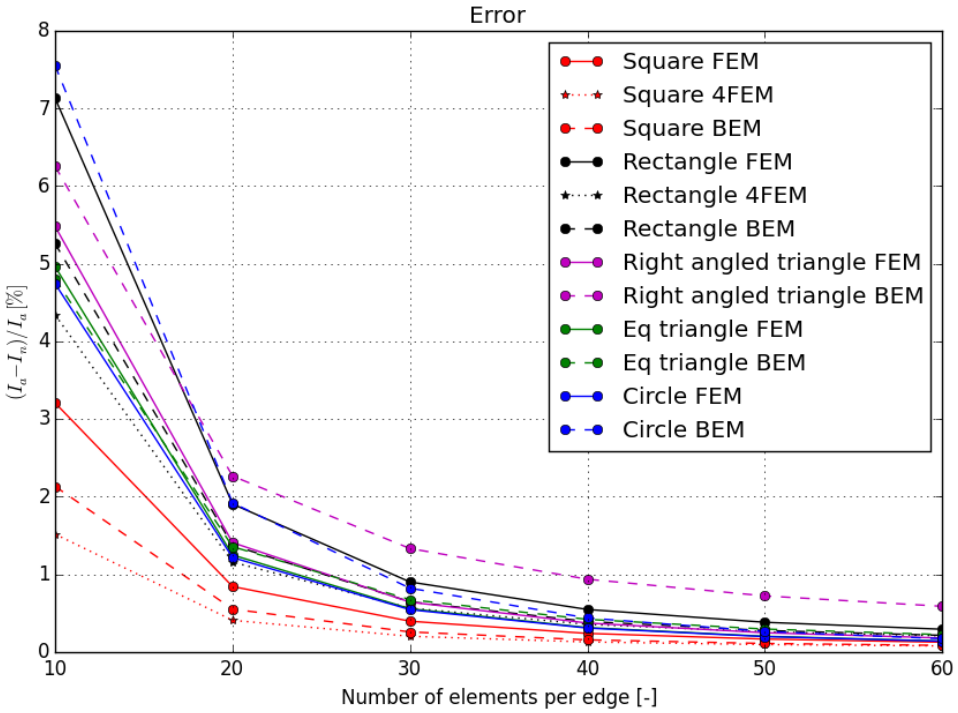


Figure 4.22: Convergence of the error

The error is computed similarly as in Eq. (4.1) where $m1$ denotes analytical value and $m2$

is equal to numerical value. The difference is less than 1 % with 40 and more elements per edge.

4.2 Application in author's software TorPy + external mesh generator

The maximum stress of the shaft with the keyseat is located in the vicinity of the radius, as expected. The range of the legend is also influenced by the boundary conditions. The detailed Fig. 4.23 shows, that the resulting maximum stress is approximately 140 MPa.

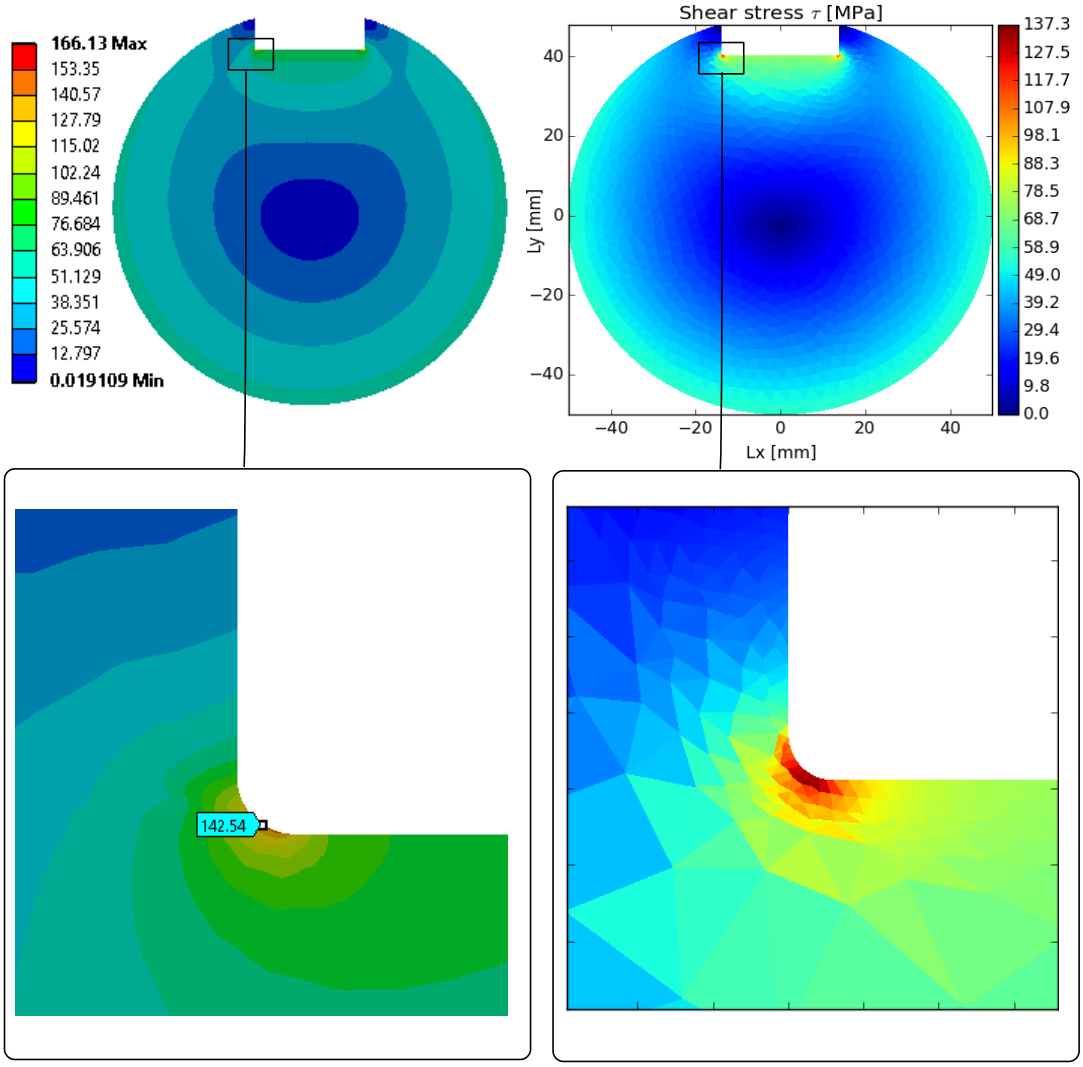


Figure 4.23: Maximum shear stress τ_{max} of the shaft with the keyseat (Ansys - TorPy)

The computation time exceeded one hour in Ansys. The TorPy needed only seven and half minute for it.

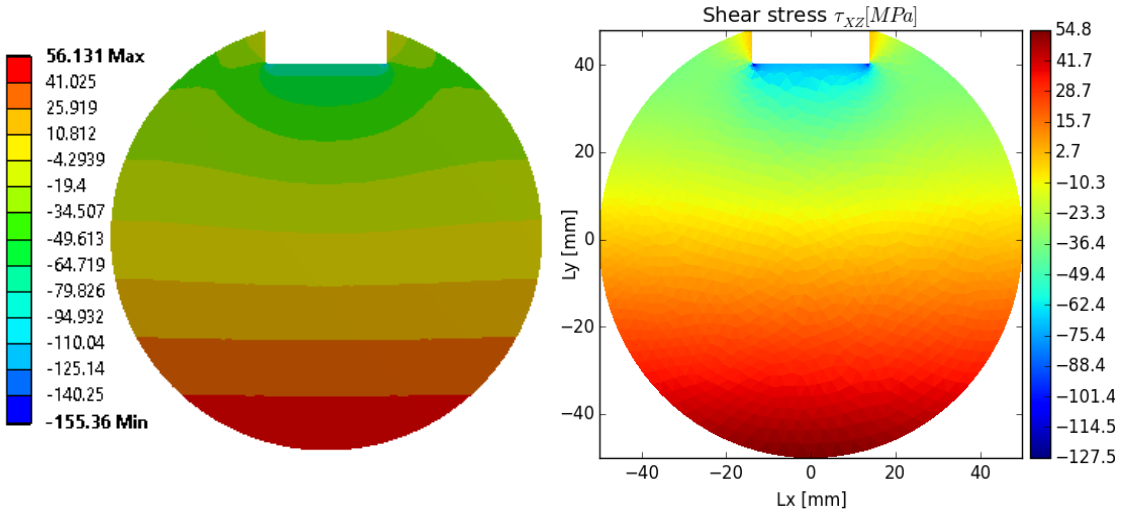


Figure 4.24: Shear stress τ_{xz} of the shaft with the keyseat (Ansys - TorPy)

The results of stress components τ_{xz} and τ_{yz} from TorPy corresponds to results from commercial software Ansys, as shown in Fig. 4.24 and 4.25.

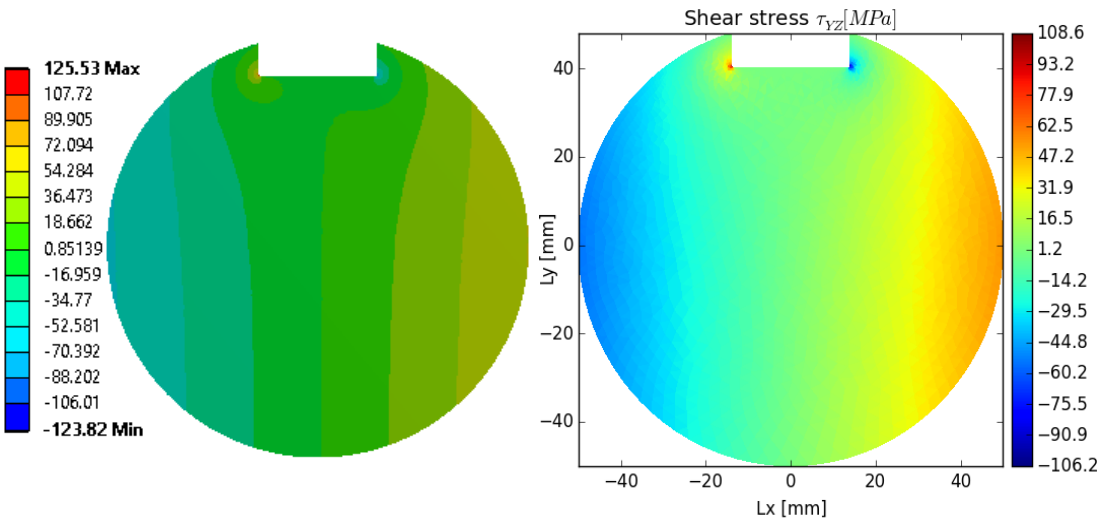


Figure 4.25: Shear stress τ_{yz} of the shaft with the keyseat (Ansys - TorPy)

Chapter 5

Conclusions

The topic of the thesis is introduced in the first chapter. The aim of the thesis is to develop an own software based on finite and boundary element method. The software is called TorPy.

Next chapter begins with the assumptions of the torsion theory. Then, it is derived the theory of torsion of circular bars and non-circular bars. The differential equation of the torsion is discretized with numerical methods, the finite element method (FEM) and the boundary element method (BEM). The end of the chapter describes the comparison of both methods.

The theory is applied on one-dimensional and two-dimensional examples using both methods, FEM and BEM. The two-dimensional cases show the problem of torsion and the base of the software TorPy.

The analytical solutions for circular, rectangular, right-angled and eqilateral triangles are introduced in Chapter 3. The own meshgenerator was developed and the principles of meshing are shown in Figures 3.5 through 3.13. The analysis of the cross-sections (see Fig. 3.4) with same volumes (weights) compares the maximum stress caused by torque. The torsion is solved by membrane analogy and the volume under membrane is computed. The volumes from FEM and BEM are compared with analytical solution. The software TorPy is able to transform mesh created in commercial software MSC Marc or MSC Patran. The mesh is used to analysis of a shaft with keyseat, as shown in Fig. 3.15.

According the shear stress (4.1), the best torsion properties meets the circular cross-section. The maximum stress is generated on the right-angled cross-section. The stress distributions of the maximum shear stress τ_{max} and shear stresses τ_{xz} and τ_{yz} of the analyzed cross-sections are shown in Figures 4.2 through 4.17. The right side of the figures represents the results from commercial software Ansys and the left side belongs to outputs from TorPy. The mesh of Ansys is coarse since the software uses quadratic elements (SOLID186), which are more accurate than linear elements employed in TorPy, it means that TorPy must use more elements for same accuracy. However, TorPy calculates two-dimensional analysis in comparison with three-dimensional analysis in Ansys. The results are almost identical and the difference is negligible.

As mentioned, the software TorPy uses membrane analogy and the membranes are shown in Figs. 4.19 and 4.20. The error of using methods is computed according to Eq. (4.1) and shown on square cross-section with 15 elements per edge in Fig. 4.21. The biggest difference is located in the corners of the cross-section, because there is the biggest gradient.

The polar moment of inertia is possible to calculate as a multiple of the volume under membrane, as derived in Section 3.1.5. The numerical values of the polar moment of inertia are shown in Table 4.1. The error of the polar moments of inertia are plotted in Fig. 4.22 and the difference is less than one percent with 40 and more elements per edge.

The analysis of the shaft with keyseat is carried out using external meshgenerator and verified with Ansys. The time for solution in Ansys exceeded one hour and the computation time from TorPy was less than eight minutes. The results are shown in Figs. 4.23 through 4.25 and it was achieved accurate results. The code of the software TorPy is attached in Appendix.

Future work

- Use higher order elements in BEM
- Illustration of the stress in BEM
- Combine FEM and BEM
- Combine triangular and quadrilateral elements in FEM
- Parametrise more types of cross-sections
- Optimize the code of TorPy
- Transformation of the mesh from free software (e.g. Salome Meca)
- Automatic application of symmetric boundary conditions
- Transform code to solve steady-state heat transfer

Acknowledgment

I would like to sincerely thank my supervisor Ing. Alexandros Markopoulos, Ph.D. for his advice and expert guidance. I also thank my family and my girlfriend for support during study and last but not least I thank to all teaching staff and colleagues for providing me with their knowledge and experiences.

List of Figures

1 Modelling procedure 20

1.1 Bar with circular cross-section 22
1.2 Deformed bar after application of a torque 22
1.3 Zoom of the small element of the bar 23
1.4 Circular cross-section with stress distribution 24
1.5 Deformation of circular bar and warping of a rectangular bar 25
1.6 Torques acting on a non-circular bar 26
1.7 Cross-section before (black) and after deformation (dashed brown) 26
1.8 Zoom of the triangle 0PA 28
1.9 Traction vector on tetrahedron 31
1.10 Establishment of boundary conditions and torque 32
1.11 Membrane ζ and function ϕ 35
1.12 The boundary 37
1.13 Triangular element 39
1.14 Element shape functions 41
1.15 Example of a sparse matrix 42
1.16 Sparse matrix notation 43
1.17 Rectangular element 44
1.18 Mapping (transformation) into quadrilateral isoparametric element 45
1.19 Membrane and its discretization 48
1.20 CGM algorithm 50
1.21 Domain A and boundary Γ 52
1.22 Investigation of a load point ξ on boundary Γ 55
1.23 Approximations of geometry and unknown field 57
1.24 One-dimensional shape functions 58
1.25 Normals of the elements 60
1.26 FEM vs BEM procedure[13] 63

2.1 Exact function and approximation 66
2.2 Comparison of exact and numerical solution 70

2.3 2D example . . . 71
2.4 2D example - FEM . . . 72
2.5 The stiffness matrix . . . 74
2.6 2D example - BEM . . . 75
2.7 BEM vectors . . . 76
2.8 BEM vectors . . . 79

3.1 Rectangular cross-section . . . 82
3.2 Triangular cross-sections . . . 82
3.3 Beam . . . 83
3.4 Parametric study . . . 85
3.5 Rectangle - finite element mesh . . . 86
3.6 Mesh of rectangle cross section . . . 87
3.7 Rectangle - finite element mesh . . . 87
3.8 Right-angled triangle - finite element mesh . . . 88
3.9 Mesh of right-angled triangle cross section . . . 88
3.10 Right-angled triangle - finite element mesh . . . 89
3.11 Mesh of equilateral triangle cross section . . . 89
3.12 Circle - finite element mesh . . . 90
3.13 Mesh of circle cross section (one quarter) . . . 90
3.14 Volume calculation . . . 93
3.15 Key groove shaft . . . 94
3.16 Boundary conditions . . . 94
3.17 Mesh . . . 95
3.18 Comparison of meshes from Ansys and TorPy . . . 96

4.1 Convergence . . . 98
4.2 Maximum shear stress τ_{max} of the circle (Ansys - TorPy) . . . 98
4.3 Shear stress τ_{xz} of the circle (Ansys - TorPy) . . . 99
4.4 Shear stress τ_{yz} of the circle (Ansys - TorPy) . . . 99
4.5 Maximum shear stress τ_{max} of the square (Ansys - TorPy) . . . 100
4.6 Shear stress τ_{xz} of the square (Ansys - TorPy) . . . 100
4.7 Shear stress τ_{yz} of the square (Ansys - TorPy) . . . 101
4.8 Maximum shear stress τ_{max} of the equilateral rectangle (Ansys - TorPy) . . . 101
4.9 Shear stress τ_{xz} of the rectangle (Ansys - TorPy) . . . 102
4.10 Shear stress τ_{yz} of the rectangle (Ansys - TorPy) . . . 102
4.11 Maximum stress τ_{max} of the rectangle and square with quadrilateral elements 103
4.12 Maximum shear stress τ_{max} of the equilateral triangle (Ansys - TorPy) . . . 103
4.13 Shear stress τ_{xz} of the equilateral triangle (Ansys - TorPy) . . . 104

4.14 Shear stress τ_{yz} of the equilateral triangle (Ansys - TorPy) 104
4.15 Maximum shear stress τ_{max} of the right-angled triangle (Ansys - TorPy) . . . 105
4.16 Shear stress τ_{xz} of the right-angled triangle (Ansys - TorPy) 105
4.17 Shear stress τ_{yz} of the right-angled triangle (Ansys - TorPy) 106
4.18 Distribution of stress along the length - Ansys 106
4.19 Membranes over one-quarter of circle and square cross-sections 107
4.20 Membranes over equilateral triangle and right-angled triangle cross-sections . 108
4.21 Error 109
4.22 Convergence of the error 110
4.23 Maximum shear stress τ_{max} of the shaft with the keyseat (Ansys - TorPy) . . 111
4.24 Shear stress τ_{xz} of the shaft with the keyseat (Ansys - TorPy) 112
4.25 Shear stress τ_{yz} of the shaft with the keyseat (Ansys - TorPy) 112

List of Tables

1.1 Membrane analogy 35
1.2 Comparison of FEM and BEM 62

3.1 Coefficients 82
3.2 The task 91

4.1 Analytical and numerical value of the polar moment of inertia 110

Bibliography

[1] AUSTRELL, Per Erik, DAHLBLOM, Ola, et al. *CALFEM A finite element toolbox Version 3.4*. http://www.solid.lth.se/fileadmin/hallfasthetslara/utbildning/kurser/FHL064_FEM/calfem34.pdf, 2017-05-14, Lund University, 2004.

[2] BATHE, Klaus-Jürgen. *Finite element procedures*. Upper Saddle River: Prentice Hall, 1996. ISBN 0-13-301458-4.

[3] BEARDMORE, Roy. *Torsion Equations*. http://www.roymech.co.uk/Useful_Tables/Torsion/Torsion.html,2017-04-19. 2010.

[4] BECKER, Adib. *The boundary element method in engineering: a complete course*. McGraw-Hill Companies, 1992. ISBN 0077074394.

[5] BEER, Gernot, SMITH Ian and DUENSER, Christian. *The Boundary Element Method with Programming*. SpringerWienNewYork, 2008. ISBN 97-3-211-71574-1.

[6] BITTNAR, Zdeněk and ŠEJNOHA, Jiří. *Numerické metody mechaniky 1*. ČVUT, 1992. ISBN 80-01-00855-X.

[7] BITTNAR, Zdeněk and ŠEJNOHA, Jiří. *Numerické metody mechaniky 2*. ČVUT, 1992. ISBN 80-01-00901-7.

[8] BREBBIA, Carlos Alberto and DOMINGUEZ, Jose. *Boundary elements: an introductory course*. WIT press, 1994. ISBN 80-01-00901-7.

[9] CAMPANILE, A., MANDARINO, M., PISCOPO, V. and PRANZITELLI, A. On the Exact Solution of Non-Uniform Torsion for Beams with Axial Symmetric Cross-Section *World Acad. Sci. Eng. Technol.* **3**(31), 36-45 , 2009.

[10] COOK, R., MALKUS, D., PLESHA, M. and WITT, R. *Concepts and applications of finite element analysis*. John Wiley & Sons Inc, 2002.

[11] FENNER, Roger. *Boundary Element Methods for Engineers: Part I*. https://thybowwh.files.wordpress.com/2014/09/boundary-element-methods-for-engineers-part-i.pdf. 2014, ISBN: 978-87-403-0732-0.

[12] FUSEK, Martin and HALAMA, Radim.*MKP a MHP*. Západočeská univerzita v Plzni, Vysoká škola báňská–Technická univerzita Ostrava, 2011. http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/metoda_konecnych_prvku_a_hranicnich_prvku.pdf.

[13] GAUL, Lothar, KÖGL, Martin and WAGNER, Marcus.*Boundary element methods for engineers and scientists: an introductory course with advanced topics*. Springer Science & Business Media, 2013.

[14] HÁJEK, Emanuel, REIF, Pavel and VALENTA, František.*Pružnost a pevnost 1*. Státní nakladatelství technické literatury, 1988.

[15] HESTENES, Magnus Rudolph and STIEFEL, Eduard.*Methods of conjugate gradients for solving linear systems*. NBS, 1952.

[16] HÖSCHL, Cyril. *Pružnost a pevnost ve strojnictví*. Státní nakladatelství technické literatury, 1971.

[17] KOZUBEK, T., BRZOBOHATỲ, T., HAPLA, V., JAROŠOVÁ, M. and MARKOPOULOS, A. *Lineární algebra s Matlabem*. Matematika pro inženỳry, 2012.

[18] KRUTINA, Jaroslav. *Sbírka vzorcú z pružnosti a pevnosti*. SNTL, 1962.

[19] KUBA, František. *Teorie pružnosti a vybrané aplikace*. SNTL, 1977.

[20] LAFORCE, Tara. *Boundary Element Method Course Notes*. Stanford, 2017. `https://web.stanford.edu/class/energy281/BoundaryElementMethod.pdf`, 2006.

[21] LEINVEBER, Jan and VÁVRA, Pavel. *Strojnické tabulky. 4. doplněné vydání*. Úvaly: ALBRA, 2008. ISBN 978-80-7361-051-7.

[22] LENERT, Jiří. *Pružnost a pevnost I*. Vysoká škola báňská-Technická univerzita, 2009. ISBN 80-7078-392-3.

[23] LENERT, Jiří. *Pružnost a pevnost II*. Vysoká škola báňská-Technická univerzita, 2009. ISBN 80-7078-572-1.

[24] MIKULČÁK Jiří, et al. *Matematické, fyzikální*. SPN, Praha, 1988. ISBN 80-85849-84-4.

[25] OTTOSEN, Niels Saabye and PETERSSON, Hans and SAABYE, Niels. *Introduction to the finite element method*. Prentice Hall International, 1992. ISBN 0-13-473877-2.

[26] BREDDY, Junuthula Narasimha. *An introduction to continuum mechanics*. Cambridge university press, 2007. ISBN 9780521870443.

[27] SAAD, Yousef. *Iterative methods for sparse linear systems*. SIAM, 2000. ISBN 0898715342.

[28] SOKOLNIKOFF, Ivan Stephen, SPRECHT, Robert Dickerson, et al. *Mathematical theory of elasticity*. Hoboken, McGraw-Hill New York, 1956.

[29] SAPOUNTZAKIS, Evangelos and MOKOS, Vasilios. Warping shear stresses in nonuniform torsion by BEM. *Springer*. 2003, **2**(20), 131-142.

[30] TIMOSHENKO, Stephen and GOODIER J. *Theory of Elasticity*. McGraw-Hill Publishing Company, 1970. ISBN13: 978-0070647206.

[31] TUTIASHVILI, Tea. *Solution of Torsion of Beams with Non-Circular Cross-Sections in Python.* Ostrava, The Czech Republic, 2016. Diploma thesis. VSB-Technical University of Ostrava.

[32] ZAPLETAL, Jan. Aplikace metody hraničních prvků na řešení Dirichletovy-Neumanovy okrajové úlohy. Ostrava, The Czech Republic, 2009. Diploma thesis. VSB-Technical University of Ostrava.

Notes

Table of Contents for Appendix

Appendix	Number of pages
A Diras's delta function	1
B Green-Gauss theorem	1
C Gaussian quadrature	2
D FEM codes	26
E BEM codes	7

Listings

5.1 Mesh - circle . . . 1
5.2 Mesh - rectangle triangular . . . 2
5.3 Mesh - rectangle quadrilateral . . . 3
5.4 Mesh - right angled triangle . . . 3
5.5 Mesh - equilateral triangle . . . 5
5.6 Mesh selector . . . 7
5.7 Lagrange multipliers . . . 9
5.8 Conjugate gradient method . . . 10
5.9 Analytical solutions . . . 11
5.10 FEM3 Main code . . . 13
5.11 FEM4 Main code . . . 19
5.12 Gaussian integration . . . 23
5.13 Stress FEM4 . . . 24
5.14 Paraview transformation . . . 25
5.15 External mesh generator . . . 26
5.16 BEM Main code . . . 1
5.17 Mesh FEM to BEM . . . 6

A Dirac's delta function

The Dirac's delta is a singularity function given as

$$\delta(x,\xi) = \begin{cases} 0, x \neq \xi \\ \infty, x = \xi \end{cases} \tag{5.1}$$

and

$$\int_{-\infty}^{+\infty} \delta(x,\xi)\mathrm{d}x = 1 \tag{5.2}$$

or other notation, e.g. according to [25], is given as

$$\int_{-\infty}^{+\infty} \delta(x-\xi)\mathrm{d}x = 1 \ . \tag{5.3}$$

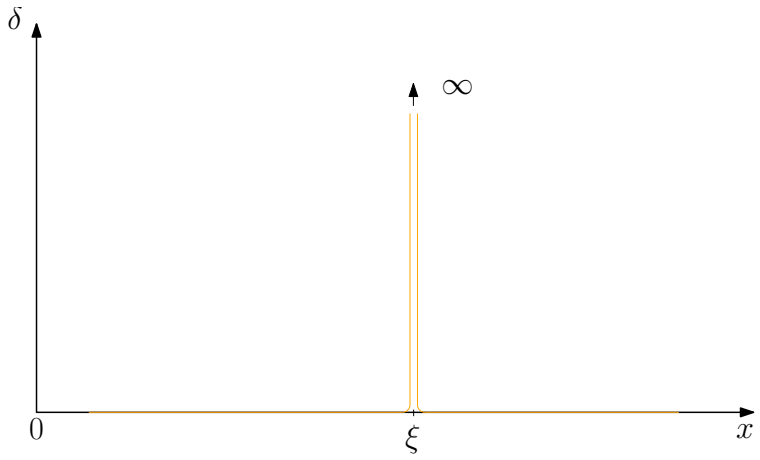


Figure 5.1: Dirac's delta function

Equation (5.1) or its plot in Figure (5.1) show that the contribution to the integral (5.3) is very close to the point ξ. It means that the integral may be rewritten as

$$\int_{\xi-}^{\xi+} \delta(x,\xi)\mathrm{d}x = 1 \tag{5.4}$$

where $\xi-$ and $\xi+$ are values slightly larger and slightly smaller than ξ, respectively. Other property of Dirac's delta function is called 'sifting property', i.e.

$$\int_{-\infty}^{+\infty} f(x)\delta(x,\xi)\mathrm{d}x = f(\xi) \ . \tag{5.5}$$

It is used for an evaluation of point sources.

B Green-Gauss theorem

Suppose a scalar field u which depends on coordinates x and y, i.e. $u = u(x, y)$. The derivative (gradient) of the quantity may be written as

$$\nabla u = \begin{pmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \end{pmatrix} \quad . \tag{5.6}$$

Assume another function $v = v(x, y)$ and integral as

$$\iint_A v \, \mathrm{div}(\nabla u) dA \quad . \tag{5.7}$$

The Green-Gauss theorem is defined as

$$\iint_A v \, \mathrm{div}(\nabla u) \, \mathrm{d}A = \oint_\Gamma v (\nabla u)^\top \mathbf{n} \, \mathrm{d}\Gamma - \iint_A (\nabla v)^\top (\nabla u) \, \mathrm{d}A \quad . \tag{5.8}$$

In accordance with Eq. (1.112), the Equation (5.8) may be rewritten as

$$\iint_A v \, \mathrm{div}(\nabla u) \, \mathrm{d}A = \oint_\Gamma v \frac{\partial u}{\partial n} \, \mathrm{d}\Gamma - \iint_A (\nabla v)^\top (\nabla u) \, \mathrm{d}A \quad . \tag{5.9}$$

Eqs. (5.8) and (5.9) are used in Chapter 1. The detailed derivation of the Green-Gauss theorem is written in [25].

C Gaussian quadrature

An analytical solution of an integral is not always possible and the integral has to be evaluated numerically. The numerical integration is implemented in all commercial software and the most used method is so-called Gaussian quadrature or Gaussian integration. For illustration of this method, it is convenient to consider integral I as

$$I = \int_{-1}^{1} f(x)\mathrm{d}x \tag{5.10}$$

and its approximation is given by

$$I \approx \sum_{i=1}^{n} f(\omega_i) H_i \tag{5.11}$$

where $f(x)$ denotes an arbitrary function, ω_i is an arbitrary point inside domain (-1,1), H_i represents a weight and n is the number of integration points (see Fig. 5.2). Assume a real function given as

$$f(x) = x^4 + 2 \tag{5.12}$$

and the exact integral in the interval (-1,1) is given as

$$I = \int_{-1}^{1} (x^4 + 2)\,\mathrm{d}x = \frac{22}{5} = 4.400\ . \tag{5.13}$$

The simplest method for the numerical integration is called Rectangle method or Midpoint rule where the original domain is divided by number of points and each interval has a same length. The integration is illustrated in Fig. 5.2a).

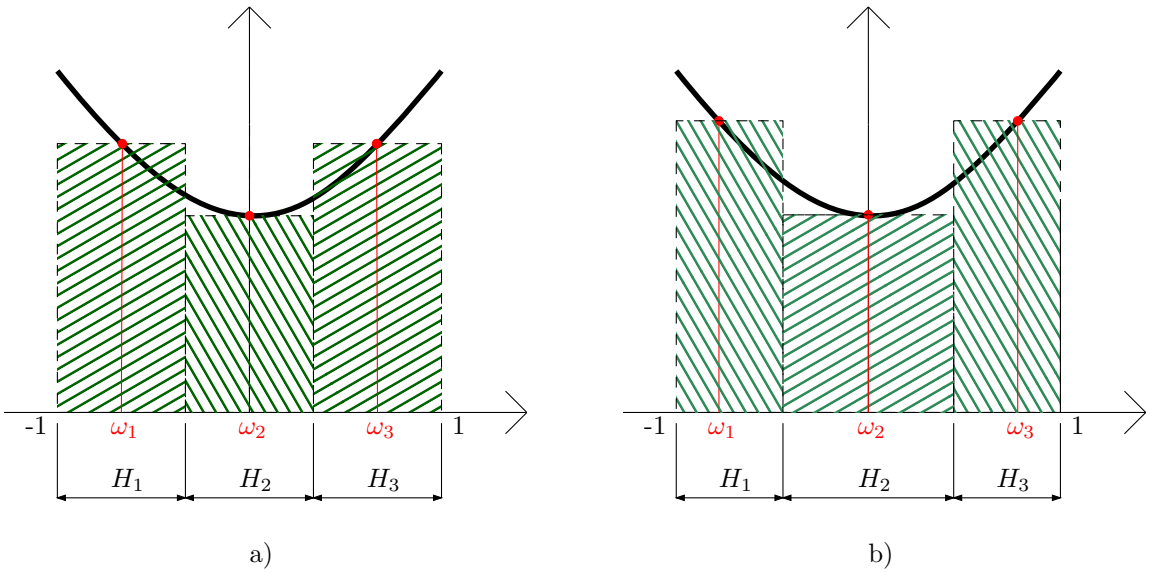


Figure 5.2: Numerical integration

Assume three points inside domain and the general formula for the approximation of the integral may be written as

$$I = f(\omega_1)H_1 + f(\omega_2)H_2 + f(\omega_3)H_3 \tag{5.14}$$

Points and weights for the midpoint rule are shown in Table 5.1.

	Midpoint rule	Gauss integration
ω_1	$-0.\overline{66}$	-0.774596669241483
ω_2	0	0
ω_3	$0.\overline{66}$	0.774596669241483
H_1	$0.\overline{66}$	$0.\overline{55}$
H_2	$0.\overline{66}$	$0.\overline{88}$
H_3	$0.\overline{66}$	$0.\overline{55}$

Table 5.1: Numerical integration - example

Therefore, from Eq. (5.14) implies that

$$I = (-0.\overline{66}^4 + 2) \cdot 0.\overline{66} + (0^4 + 2) \cdot 0.\overline{66} + (0.\overline{66}^4 + 2) \cdot 0.\overline{66} = 4.263 \tag{5.15}$$

and the result is not exact. More accurate solution provides Gaussian quadrature. The method differs in positions of integration points and the weights, as shown in Fig 5.2b). Gaussian rule provides an exact solution of a polynomial of the order 2n-1 for n integration points. The positions of the integration points and the weights for n points are written in Table 5.2. The integral of the function (5.12) gives with the integration points and weights from Tab. 5.1

$$I = (-0.7746^4 + 2) \cdot 0.\overline{55} + (0^4 + 2) \cdot 0.\overline{88} + (0.7746^4 + 2) \cdot 0.\overline{55} = 4.400 \tag{5.16}$$

and this result is exact. A derivation of Gaussian quadrature and detailed information can be find, for example, in [25].

n	ω_i	H_i
1	0.000000000000000	2.00
2	$\pm$0.577350269189626	1.00
3	0.000000000000000	$0.\overline{88}$
	$\pm$0.774596669241483	$0.\overline{55}$

Table 5.2: Gaussian points

D FEM Codes

Listing 5.1: Mesh - circle

```
from __future__ import division
import numpy as np
#The function generates circular mesh
def mesh(row,L):
    #functions from interpolations
    def elem(r):
        return 2*r-1
    def node(r):
        return 0.5*r**2+1.5*r+1
    def first_col(r):
        return 0.5*(r**2+r)
    def edge2(r):
        return 0.5*r**2+1.5*r
    ran=[]
    for i in range(row):
        ran.append(first_col(i))
    tria_up=ran[-1]
    #number of elements according to rows
    n_e = 0
    for i in range(row):
        n_e += elem(i+1)
    #number of nodes according to rows
    n_nod = int(node(row))
    elements = np.zeros((n_e,3),int)
    coordinates= np.zeros((n_nod,2))
    isOnEdge = np.zeros(n_nod,dtype=np.bool)
    tria_low = n_e-int(tria_up)
    for i in range(tria_low):
        elements[i,0]=i
    cnt = 0
    cnt1 = 0
    cnt2 = 0
    cnt3 = 0
    first=[]
    for i in range(row+1):
        first.append(first_col(i))
    first.remove(0)
    while cnt <row:
        for i in range(cnt+1):
            if i ==0:
                elements[cnt1,1]=int(first[cnt2])
                elements[cnt1,2]=int(first[cnt2])+1
                cnt2+=1
            if i>0:
                elements[cnt1,1]=elements[cnt1-1,2]
                elements[cnt1,2]=elements[cnt1-1,2]+1
                elements[tria_low+cnt3,1]= cnt1+cnt+1
```

```
                elements[tria_low+cnt3,0]= cnt1-1
                elements[tria_low+cnt3,2]= cnt1
                cnt3+=1
            cnt1+=1
        cnt+=1
    cnt = 0
    cnt1 = 0
    while cnt <row+1:
        for i in range(cnt+1):
            if cnt>0:
                coordinates[cnt1,1]= cnt*L/row*np.sin(i*np.pi/(2*cnt))
                coordinates[cnt1,0]= cnt*L/row*np.cos(i*np.pi/(2*cnt))
            if cnt == row:
                isOnEdge[cnt1]=True
            cnt1+=1
        cnt+=1
    el_type = "t"       #element type
    sf =4               #symmetry factor
    return coordinates, elements,n_nod,isOnEdge,el_type,sf
```

Listing 5.2: Mesh - rectangle triangular

```
from __future__ import division
import numpy as np
#The function generates rectangular mesh - three nodes
def mesh(n1,n2,L1,L2):
    h1 = L1/n1
    h2 = L2/n2
    n_nod = (n1+1)*(n2+1)
    nel = n1*n2*2
    elements = np.zeros((nel,3),dtype = np.int32)
    coordinates = np.zeros((n_nod,2),dtype=np.float32)
    cnt = 0
    nx = n1 + 1
    ny = n2 + 1
    for j in range(n2):
        for i in range (n1):
            elements[cnt,:] = np.array([i,i+1,nx+i+1])+nx*j
            elements[cnt+1,:] = np.array([i,nx+i+1,nx+i])+nx*j
            cnt +=2
    cnt = 0
    isOnEdge = np.zeros(n_nod,dtype=np.bool)
    for j in range(ny):
        for i in range(nx):
            coordinates[cnt,:] = np.array([i*h1, j*h2])
            if (i==0 or j==0 or i==n1 or j == n2):
                isOnEdge[cnt]=True
            cnt += 1
    el_type = "t"    #element type
    sf = 1           #symmetry factor, possible to make a symmetry two times
    return coordinates, elements,n_nod,isOnEdge,el_type,sf
```

Listing 5.3: Mesh - rectangle quadrilateral

```
from __future__ import division
import numpy as np
import matplotlib.pylab as plt
#The function generates rectangular mesh - four nodes
def mesh(n1,n2,L1,L2):
    h1 = L1/n1
    h2 = L2/n2
    n_el = n1*n2
    n_nod = (n1+1)*(n2+1)
    elements = np.zeros((n_el,4),dtype=np.int)
    coordinates = np.zeros((n_nod,2),dtype=np.float32)
    cnt = 0
    cnt1 = 0
    for i in range (n2):
        for j in range (n1):
            elements[cnt,:]=np.array([j+cnt1,j+cnt1+1,j+cnt1+n1+2,j+cnt1+n1+1])
            cnt+=1
        cnt1+=n1+1
    cnt = 0
    isOnEdge = np.zeros(n_nod,dtype=np.bool)
    for j in range(n2+1):
        for i in range(n1+1):
            coordinates[cnt,:] = np.array([i*h1, j*h2])
            if (i==0 or j==0 or i==n1 or j == n2):
                isOnEdge[cnt]=True
            cnt += 1
    el_type = "q"     #element type
    return coordinates, elements,n_nod,isOnEdge,el_type
```

Listing 5.4: Mesh - right angled triangle

```
from __future__ import division
import numpy as np
#The function generates right-angled triangular mesh
def mesh(row,L):
    #functions from interpolations:
    def elem(r):
        return 2*r-1
    def node(r):
        return 0.5*r**2+1.5*r+1
    def first_col(r):
        return 0.5*(r**2+r)
    def edge2(r):
        return 0.5*r**2+1.5*r
    ran=[]
    for i in range(row):
        ran.append(first_col(i))
    tria_up=ran[-1]
    #number of elements according to rows
    n_e = 0
```

```
    for i in range(row):
        n_e += elem(i+1)
    #number of nodes according to rows
    n_nod = int(node(row))
    elements = np.zeros((n_e,3),int)
    coordinates= np.zeros((n_nod,2))
    tria_low = n_e-int(tria_up)
    for i in range(tria_low):
        elements[i,0]=i
    cnt = 0
    cnt1 = 0
    cnt2 = 0
    cnt3 = 0
    first=[]
    for i in range(row+1):
        first.append(first_col(i))
    first.remove(0)
    while cnt <row:
        for i in range(cnt+1):
            if i ==0:
                elements[cnt1,1]=int(first[cnt2])
                elements[cnt1,2]=int(first[cnt2])+1
                cnt2+=1
            if i>0:
                elements[cnt1,1]=elements[cnt1-1,2]
                elements[cnt1,2]=elements[cnt1-1,2]+1
                elements[tria_low+cnt3,1]= cnt1+cnt+1
                elements[tria_low+cnt3,0]= cnt1-1
                elements[tria_low+cnt3,2]= cnt1
                cnt3+=1
            cnt1+=1
        cnt+=1
    cnt = 0
    cnt1 = 0
    while cnt <row+1:
        for i in range(cnt+1):
            coordinates[cnt1,1]= i*L/row
            cnt1+=1
        cnt+=1
    cnt = 0
    cnt1 = 0
    while cnt <row+1:
        for i in range(cnt+1,0,-1):
            coordinates[cnt1,0]=(i-1)*L/row
            cnt1+=1
        cnt+=1
    isOnEdge = np.zeros(n_nod,dtype=np.bool)
    for i in range(row):
        isOnEdge[0]=True
        isOnEdge[-(i+1)]=True
        isOnEdge[first_col(i+1)]=True
        isOnEdge[edge2(i+1)]=True
```

```
    el_type = "t"      #element type
    sf = 1             #symmetry factor
    return coordinates, elements,n_nod,isOnEdge,el_type,sf
```

Listing 5.5: Mesh - equilateral triangle

```
from __future__ import division
import numpy as np
#The function generates equilateral tringle mesh
def mesh(row,L):
    #functions from interpolations:
    def elem(r):
        return 2*r-1
    def node(r):
        return 0.5*r**2+1.5*r+1
    def first_col(r):
        return 0.5*(r**2+r)
    def edge2(r):
        return 0.5*r**2+1.5*r
    ran=[]
    for i in range(row):
        ran.append(first_col(i))
    tria_up=ran[-1]
    #number of elements according to rows
    n_e = 0
    for i in range(row):
        n_e += elem(i+1)
    #number of nodes according to rows
    n_nod = int(node(row))
    elements = np.zeros((n_e,3),int)
    coordinates= np.zeros((n_nod,2))
    tria_low = n_e-int(tria_up)
    for i in range(tria_low):
        elements[i,0]=i
    cnt = 0
    cnt1 = 0
    cnt2 = 0
    cnt3 = 0
    first=[]
    for i in range(row+1):
        first.append(first_col(i))
    first.remove(0)
    while cnt <row:
        for i in range(cnt+1):
            if i ==0:
                elements[cnt1,1]=int(first[cnt2])
                elements[cnt1,2]=int(first[cnt2])+1
                cnt2+=1
            if i>0:
                elements[cnt1,1]=elements[cnt1-1,2]
                elements[cnt1,2]=elements[cnt1-1,2]+1
```

```
                elements[tria_low+cnt3,1]= cnt1+cnt+1
                elements[tria_low+cnt3,0]= cnt1-1
                elements[tria_low+cnt3,2]= cnt1
                cnt3+=1
            cnt1+=1
        cnt+=1
    cnt = 0
    cnt1 = 0
    while cnt <row+1:
        for i in range(cnt+1):
            coordinates[cnt1,1]= i*L/row*np.cos(np.pi/6)
            coordinates[cnt1,0]=(cnt-0.5*i)*L/row
            cnt1+=1
        cnt+=1
    isOnEdge = np.zeros(n_nod,dtype=np.bool)
    for i in range(row):
        isOnEdge[0]=True
        isOnEdge[-(i+1)]=True
        isOnEdge[first_col(i+1)]=True
        isOnEdge[edge2(i+1)]=True
    el_type = "t"  #element type
    sf = 1         #symmetry factor
    return coordinates, elements,n_nod,isOnEdge,el_type,sf
```

Listing 5.6: Mesh selector

```
import numpy as np
#This function selects the mesh and its dimensions, if available
def mesh(name):
    if name=='rectangle':
        import rectangle as mg
        a = input('Insert size a (Bigger size): ') #bigger size!
        b = input('Insert size b: ')   #smaller size
        ny = input('Insert number of element at edge X: ')
        nx = input('Insert number of element at edge Y: ')
        dim = np.array([a,b,nx,ny])
        coordinates, elements,n_no,isOnEdge,el_type,sf = mg.mesh(nx,ny,a,b)

    elif name=='right angle triangle':
        import right_ang_trian as mg
        L =  input('Insert a length of leg(odvesna): ')
        row =  input('Insert number of rows: ')
        dim = L
        coordinates, elements,n_no,isOnEdge,el_type,sf = mg.mesh(row,L)

    elif name=='equilateral triangle':
        import equilat_trian as mg
        L =  input('Insert a length of edge: ')
        row =  input('Insert number of rows: ')
        dim = L
        coordinates, elements,n_no,isOnEdge,el_type,sf = mg.mesh(row,L)

    elif name=='circle':
        import circ as mg
        L =  input('Insert a radius: ')
        row =  input('Insert number of elements per radius: ')
        dim = L
        coordinates, elements,n_no,isOnEdge,el_type,sf=mg.mesh(row,L)

    elif name=='experiment':
        elements = np.array([[0,1,2],[1,3,2],[3,4,2],[4,0,2]])
        coordinates = np.array([[0,0],[0.05,0],[0.025,0.025],[0.05,0.05],[0,0.05]])
        n_no = 5
        isOnEdge = np.array([False,True,False,True,True])
        el_type = 't'
        sf = 4
        dim = None

    elif name=='patran':
        import sys
        sys.path.append('../../PATRAN_EXPORT_MESH')
        import patran_reader_func2D as msh
        coordinates, elements= msh.mesh('drazka.out')
        bound = np.zeros([3*elements.shape[0],2],dtype=np.int)
        for i in range(elements.shape[0]):
            bound[i,:]=elements[i,0:2]
```

```
            bound[i+elements.shape[0],:]=elements[i,1:3]
            bound[i+2*elements.shape[0],:]=elements[i,0:3:2]
            if bound[i,0]>bound[i,1]:
                temp=bound[i,0]
                bound[i,0]=bound[i,1]
                bound[i,1]=temp
            if bound[i+elements.shape[0],0]>bound[i+elements.shape[0],1]:
                temp=bound[i+elements.shape[0],0]
                bound[i+elements.shape[0],0]=bound[i+elements.shape[0],1]
                bound[i+elements.shape[0],1]=temp
            if bound[i+2*elements.shape[0],0]>bound[i+2*elements.shape[0],1]:
                temp=bound[i+2*elements.shape[0],0]
                bound[i+2*elements.shape[0],0]=bound[i+2*elements.shape[0],1]
                bound[i+2*elements.shape[0],1]=temp

        pos_del = []
        for i in range(3*elements.shape[0]):
            for j in range(3*elements.shape[0]):
                if bound[i,0]==bound[j,0] and bound[i,1]==bound[j,1]:
                    if i!=j:
                        pos_del.append(i)
                        pos_del.append(j)

        bound = np.delete(bound,pos_del,0)
        bound = np.unique(np.sort(np.reshape(bound,2*bound.shape[0])))
        isOnEdge = np.in1d(np.arange(coordinates.shape[0]), bound)
        n_no = coordinates.shape[0]
        sf = 1
        el_type = 't'
        dim = None
    return coordinates,elements,n_no,isOnEdge,el_type,sf,dim
```

Solvers

Listing 5.7: Lagrange multipliers

```
import numpy as np
import pylab as plt
from scipy import sparse
from scipy.sparse.linalg import dsolve
#This function uses Lagrange multipliers and Gauss elimination method to solve
    system of equations from FEM
def lagrange(BC,n_no,K_sparse_,f):

    #LAGRANGE MULTIPLIER MATRICES PREPARATION
    non_zero = len(plt.find(BC)) #Number of non-zero members
    I_lag = np.arange(0,non_zero ,1)
    J_lag = plt.find(BC)
    V_lag = np.ones(non_zero, dtype = np.float64)
    B_lag = sparse.csc_matrix((V_lag,(I_lag,J_lag)),shape=(non_zero,n_no)).tocsc()
        # Matrix B
    B_lag_ver = sparse.csc_matrix((V_lag,(I_lag,J_lag)),shape=(non_zero,n_no+
        non_zero )).tocsc() #Matrix B vertical
    K_sp = sparse.hstack([K_sparse_,sparse.csc_matrix.transpose(B_lag)]).tocsc() #
        Binding
    K_lag = sparse.vstack([K_sp,B_lag_ver]).tocsc() #Stiffness matrix
    C_lag = np.zeros(non_zero ,dtype = np.int) #Zero vector
    F_lag = np.concatenate((f,C_lag),axis=0) #Right-hand side

    #LAGRANGE MULTIPLIER
    phi_lag=dsolve.spsolve(K_lag,F_lag) #Lagrange
    phi_l = phi_lag[0:n_no:]
    return phi_l
```

Listing 5.8: Conjugate gradient method

```
import numpy as np
from scipy import sparse
#This function uses Conjugate gradient method for solving FE sparse matrices
def my_dot(A,x):
    return sparse.csc_matrix.dot(A,x)

def my_cg(A,b):
    n   = A.shape[0]  #dimension of A matrix
    iA  = 1.0/A.diagonal() #preconditioner
    P   = sparse.spdiags(iA,0,n,n).tocsc()
    tol = 1e-4*np.linalg.norm(b)
    x   = np.zeros((n),dtype=np.float64)
    g   = sparse.csc_matrix.dot(A,x) - b
    z   = sparse.csc_matrix.dot(P,g)
    p   = z
    it  = 0
    maxIt = n
    norm_grad=np.linalg.norm(g) #norm
    while (norm_grad > tol and it<maxIt):
        Ap = sparse.csc_matrix.dot(A,p)
        alpha = -np.dot(g,z)/np.dot(p,Ap)
        x = x + p*alpha
        g_before = g
        z_before = z
        g = g + Ap*alpha
        z = sparse.csc_matrix.dot(P,g)
        beta = np.dot(z,g)/np.dot(z_before,g_before)
        p = z + p*beta
        norm_grad = np.linalg.norm(g)
        it+=1
    print ('number of iteration is ',it)
    return x
```

Analytical solution

Listing 5.9: Analytical solutions

```
from __future__ import division
import numpy as np
#Analytical solution http://www.roymech.co.uk/Useful_Tables/Torsion/Torsion.html
#Functions give the analytical solutions
####### RIGHT ANGLE TRIANGLE
def a_sol_rat(T,a,Ge):
    T_max = T/(0.057042876*a**3)
    print ("Analytical result = Maximum stress is:",T_max/1e6,"MPa")
    theta = T/(0.0260958*a**4*Ge)
    print ("Analytical result = The rate of twist is:",theta,"rad/m")
    p=a/2
    I_a = 0.4175328*p**4
    return T_max,theta,I_a
######## CIRCLE
def a_sol_cir(T,r):
    T_max = 16*T/np.pi/(2*r)**3
    p=r
    I_a = 0.5*np.pi*p**4
    print ("Analytical result = Maximum stress is:",T_max/1e6,"MPa")
    return T_max, I_a
######## EQUILATERAL TRIANGLE
#Sbirka prikladu z pruznosti a pevnosti
def a_sol_eq_tria(G,T,a):
    T_max = 20*T/a**3
    print ("Analytical result = Maximum stress is:",T_max/1e6,"MPa")
    p=a/2
    I_a = np.sqrt(3)*(2*p)**4/80
    return T_max,I_a
######## RECTANGLE
def a_sol_rect(G,T,a,b,L):
    coeff = np.array([[1,1.2,1.5,2,2.5,3,4,5,10,100],\
                [0.208,0.219,0.231,0.246,0.258,0.267,0.282,0.291,0.312,0.333],\
                [0.1406,0.1661,0.1958,0.229,0.249,0.263,0.281,0.291,0.312,0.333]])
    r = a/b   #ratio
    #
        ##########################################################################################

    if r < 1:
        print ("Wrong size!")
    elif r >= 1 and r<1.1:
        k1 = coeff[1,0]
        k2 = coeff[2,0]
    elif r >= 1.1 and r<1.35:
        k1 = coeff[1,1]
        k2 = coeff[2,1]
    elif r >= 1.35 and r<1.75:
        k1 = coeff[1,2]
        k2 = coeff[2,2]
```

```
    elif r >= 1.75 and r<2.25:
        k1 = coeff[1,3]
        k2 = coeff[2,3]
    elif r >= 2.25 and r<2.75:
        k1 = coeff[1,4]
        k2 = coeff[2,4]
    elif r >= 2.75 and r<3.5:
        k1 = coeff[1,5]
        k2 = coeff[2,5]
    elif r >= 3.5 and r<4.5:
        k1 = coeff[1,6]
        k2 = coeff[2,6]
    elif r >= 4.5 and r<=5:
        k1 = coeff[1,7]
        k2 = coeff[2,7]
    elif r >= 7.5 and r<10:
            k1 = coeff[1,8]
            k2 = coeff[2,8]
    else:
        k1=k2=1/3*(1-0.630/(r))
    T_max =T/(k1*a*b**2)
    phi = T*L/(k2*a*b**3*G)          #angle of twist [rad]
    phi_deg = phi*180/np.pi          #angle of twist [deg]

    theta = phi/L                    #rate of twist  [rad/m]
    C = T/theta                      #torsional rigidity [Nm**2]
    print ("Analytical result = Maximum stress is:",T_max/1e6,"MPa")
    print ("phi degree is ",phi_deg)
    print ("Analytical result = The rate of twist is:",theta,"rad/m")
    if a==b:
        p = a/2.
        I_a = 2.25*p**4
    elif a==2*b:
        p = a/2.
        I_a = 0.4175328*p**4
    else:
        I_a = None
    return T_max,theta,I_a
```

Main Code

Listing 5.10: FEM3 Main code

```
from __future__ import print_function
from __future__ import division
from __future__ import absolute_import
from __future__ import unicode_literals
import numpy as np
import pylab as plt
import math as mt
from mpl_toolkits.mplot3d import Axes3D
import sys
from scipy import sparse
#This code gives stresses, angle of twist, etc. of arbitrary triangular mesh
##############################################################################
"""INPUTS"""
##############################################################################

#MESH
name = 'patran' # choose circle, rectangle, equilateral triangle or right angle
    triangle
sys.path.append('../mesh')

###import certain mesh
import select_mesh as msh
if name=='rectangle':
    coordinates, elements,n_no,isOnEdge,el_type,sf,dim = msh.mesh(name)
    a = dim[0]
    b = dim[1]
    nx = dim[2]
    ny = dim[3]
elif name=='circle':
    coordinates, elements,n_no,isOnEdge,el_type,sf,dim = msh.mesh(name)
    r = dim
elif name=='equilateral triangle':
    coordinates, elements,n_no,isOnEdge,el_type,sf,dim = msh.mesh(name)
    a = dim
elif name=='right angle triangle':
    coordinates, elements,n_no,isOnEdge,el_type,sf,dim = msh.mesh(name)
    a = dim
elif name=='patran':
    coordinates, elements,n_no,isOnEdge,el_type,sf,dim = msh.mesh(name)
    coordinates/=1000
elif name == 'L':
    coordinates, elements,n_no,isOnEdge,el_type,sf,dim = msh.mesh(name)
elif name == 'experiment':
    coordinates, elements,n_no,isOnEdge,el_type,sf,dim = msh.mesh(name)


#MATERIAL
E = 2.1e11            #Young's modulus of elasticity [Pa]
```

```
48 mu = 0.3              #Poisson ratio
49 Ge = E/(2*(1+mu))     #Shear modulus [Pa]
50
51 #LOADING
52 theta1 = 1e-4  #Rate of twist - arbitrary value [rad/m]
53 T = 1e4  #Torque [Nm]
54
55 #LENGTH
56 L = 1.  #[m]
57
58 #SOLVER
59 solver = 'Conjugate gradient method' # select either 'Lagrange multipliers' or '
       Conjugate gradient method'
60
61 ##############################################################################
62 """SOLUTION"""
63 ##############################################################################
64
65 f = np.zeros(n_no) #Global load vector for triangular element - scalar problem - 3
        DOF
66
67 n_e = np.shape(elements)[0] #number of elements
68
69 #SPARSE
70 I = np.zeros((9*n_e),dtype = np.int)
71 J = np.zeros((9*n_e),dtype = np.int)
72 V = np.zeros(9*n_e, dtype = np.float64)
73 i_Ke = np.zeros((9),dtype = np.int)
74 j_Ke = np.zeros((9),dtype = np.int)
75
76 Coord = np.zeros((np.shape(elements)[1]*n_e,2))
77 cnt = 0
78 for i in range(n_e):
79     for j in elements[i,:]:
80         Coord[cnt,:]=coordinates[j,:]
81         cnt+=1
82 Coord = np.reshape(Coord,(n_e,6))
83 Element = np.zeros(np.shape(Coord)[0])
84 #Area of each trianglular element
85 A_e = np.zeros(n_e)
86 #Total area of the cross section
87 A = 0
88 # Derivation of element shape function - scalar problem
89 G= np.zeros((2*n_e,3))
90 # Derivation of element shape function - scalar problem - for one triangular
       element
91 G_e = np.zeros((2,3))
92
93 #Element stiffness matrix for triangular element - scalar problem - 3 DOF
94 K_e = np.zeros((3,3))
95 #Element load vector for triangular element - scalar problem - 3 DOF
96 f_e = np.zeros(3)
```

```
97
98  #ELEMENT STIFNESS MATRIX AND LOAD VECTOR
99  cnt = 0
100 for i in range(n_e):
101     Element[i]=i
102     xi = Coord[i,0];  yi=  Coord[i,1];   xj = Coord[i,2]
103     yj=  Coord[i,3];   xk = Coord[i,4];  yk=  Coord[i,5]
104     Cor = np.array([[1,xi,yi],[1,xj,yj],[1,xk,yk]]) #coordinates
105     A_e[i]= 0.5*np.linalg.det(Cor)
106     A+=A_e[i]
107     G[2*i] = 1/(2*A_e[i])*np.array([yj-yk,yk-yi,yi-yj])
108     G[2*i+1] = 1/(2*A_e[i])*np.array([xk-xj,xi-xk,xj-xi])
109     G_e[0] = G[2*i]
110     G_e[1] = G[2*i+1]
111     K_e = A_e[i]/Ge*np.dot(np.transpose(G_e),G_e)
112     f_e = 2/3*theta1*A_e[i]*np.ones(3)
113     f[elements[i,:]]+=f_e
114     i_Ke[:]= np.concatenate((elements[i],elements[i],elements[i]),axis = 0) #
            numbering of row in stiffness matrix
115     cnt1 = 0
116     for j in range(3):
117         j_Ke[[j%3+cnt1,j%3+1+cnt1,j%3+2+cnt1]]=[elements[i,j],elements[i,j],
                elements[i,j]] #numbering of column in stiffness matrix
118         cnt1 +=2
119
120     I[cnt:9+cnt]+=i_Ke
121     J[cnt:9+cnt]+=j_Ke
122     V[cnt:9+cnt]+=np.ravel(K_e,9)   # reshape the local stiffness matrix to vector
            and add it to global vector of values
123     cnt +=9
124
125 K_sparse_ = sparse.csc_matrix((V,(I,J)),shape=(n_no,n_no)).tocsc()
126 print ("K_sparse is done")
127 ##########################################################
128 "BOUNDARY CONDITIONS"
129 ##########################################################
130 BC = isOnEdge
131
132 def funBC(I,J,V,BC,f):
133     for i in plt.find(BC):
134         cnt = 0
135         for j in range(len(I)):
136             if i ==I[j] or i ==J[j]:
137                 if I[cnt]!=J[cnt]:
138                     V[cnt]=0
139             cnt+=1
140     n= np.count_nonzero(V)
141     I_n = np.zeros((n),dtype = np.int)
142     J_n = np.zeros((n),dtype = np.int)
143     V_n = np.zeros(n)
144     #INDECES OF NONZEROS ELEMENTS IN V VECTOR
145     chs = np.asarray(np.nonzero(V))
```

```
    #NEW VECTORS
    V_n = np.take(V,chs).reshape(np.shape(chs)[1]) #Nonzero Values
    I_n = np.take(I,chs).reshape(np.shape(chs)[1]) #Nonzero indexes of rows
    J_n = np.take(J,chs).reshape(np.shape(chs)[1]) #Nonzero indexes of colums

    K_sparse = sparse.csc_matrix((V_n,(I_n,J_n)),shape=(n_no,n_no)).tocsc()

    cnt = 0
    for i in BC:
        if i==True:
            K_sparse[cnt,cnt]=1.
            f[cnt] = 0.
        cnt+=1
    return K_sparse, f

print ("BCs are done")
K_sparse,f_sparse = funBC(I,J,V,BC,f)

##############################################
"""SOLVERS"""
##############################################
sys.path.append('../solver')

#### SPARSE MATRICES SOLVERS
############ LAGRANGE MULTIPLIER
if solver=='Lagrange multipliers':
    import lagranmulti as lagmul
    phi= lagmul.lagrange(BC,n_no,K_sparse_,f_sparse)

###################### SOLVER CONJUGATE GRADIENT METHOD
elif solver=='Conjugate gradient method':
    import cgm as cgm
    phi= cgm.my_cg(K_sparse,f_sparse)

#############################################################################################
"""POST PROCESSING"""
#############################################################################################
Iu = 0.0

#Calculated torque
T1 =0       #nastrel
for i in range(n_e):
    T1 += sf*2*A_e[i]/3*np.dot(np.array([1,1,1]),phi[elements[i,:]])
    ###integral under surface
    ie = elements[i,:]
    np.sum(phi[ie])
    Iu += np.sum(phi[ie])/3.*A_e[i]
phi = phi*T/T1
print ('TRI T1 is: ',T1)
#Integral of membrane with unit right-hand-side
Iu_unit = Iu/(2*Ge*theta1)
print ('Volume under membrane is ',Iu_unit)
```

```
I_n = sf*4*Iu_unit

#Torsional rigidity
C = T1/theta1

theta =T/C    #real rate of twist
theta_deg = theta*180/np.pi

varphi = theta/L #angle of the twist in the end of beam
#Shear Stress  xz,yz [MPa]
Sigma = np.zeros((n_e,2))
for i in range(n_e):
    Sigma[i,:]=np.dot(G[2*i:2*i+2,:],phi[elements[i,:]])/1e6
        #nasobeni je kvuli prepoctu na skutecne napeti podle zadaneho krouticiho
            momentu T
Sigma[:,0] *=-1

#Shear Stress sqrt(xz**2+yz**2)
napeti = np.zeros(n_e)
for i in range(n_e):
    napeti[i] = mt.sqrt(Sigma[i,0]**2+Sigma[i,1]**2)

scale = 1000 #m to mm

#Mesh plot
x = coordinates[:,0]*scale
y = coordinates[:,1]*scale
plt.figure(figsize=(10,5))
plt.gca().set_aspect('equal') #equal axis x and y
mesh = np.zeros(n_e,dtype = np.float64)
plt.tripcolor(x, y, elements,facecolors=mesh,shading='flat', edgecolors='w')
plt.xlabel('Lx [mm]')
plt.ylabel('Ly [mm]')
plt.title('Mesh')
for i in range(n_no):
    plt.hold("on")
    plt.text(coordinates[i,0]*scale,coordinates[i,1]*scale,int(i),fontweight='bold'
        ,color='y')
plt.show()

#Maximum shear stress
plt.figure(figsize=(10,5))
plt.gca().set_aspect('equal')
plt.tripcolor(x, y, elements,facecolors=napeti,shading='flat',edgecolors='k')
v = np.linspace(min(napeti), max(napeti), 15, endpoint=True)
plt.colorbar(cmap= plt.jet, ticks = v)
plt.xlabel('Lx [mm]')
plt.ylabel('Ly [mm]')
plt.xlim(min(coordinates[:,0])*scale,max(coordinates[:,0])*scale)
plt.ylim(min(coordinates[:,1])*scale,max(coordinates[:,1])*scale)
plt.title('Shear stress $\\tau$ [MPa]')
plt.show()
```

```
#
#
#Shear stress Sigma YZ
plt.figure(figsize=(10,5))
plt.gca().set_aspect('equal')
plt.tripcolor(x, y, elements,facecolors=Sigma[:,0],shading='flat', edgecolors='k')
#    plt.tripcolor(x, y, elements,facecolors=Sigma[:,0],shading='flat')
v = np.linspace(min(Sigma[:,0]), max(Sigma[:,0]), 15, endpoint=True)
plt.colorbar(ticks = v)
plt.xlabel('Lx [mm]')
plt.ylabel('Ly [mm]')
plt.xlim(min(coordinates[:,0])*scale,max(coordinates[:,0])*scale)
plt.ylim(min(coordinates[:,1])*scale,max(coordinates[:,1])*scale)
plt.title('Shear stress $\\tau_{YZ} [MPa]$ ')
plt.show()

#Shear stress Sigma XZ
plt.figure(figsize=(10,5))
plt.gca().set_aspect('equal')
plt.tripcolor(x, y, elements,facecolors=Sigma[:,1],shading='flat', edgecolors='k')
#    plt.tripcolor(x, y, elements,facecolors=Sigma[:,1],shading='flat')
v = np.linspace(min(Sigma[:,1]), max(Sigma[:,1]), 15, endpoint=True)
plt.colorbar(cmap=plt.cm.jet,ticks = v)
plt.xlabel('Lx [mm]')
plt.ylabel('Ly [mm]')
plt.xlim(min(coordinates[:,0])*scale,max(coordinates[:,0])*scale)
plt.ylim(min(coordinates[:,1])*scale,max(coordinates[:,1])*scale)
plt.title('Shear stress $\\tau_{XZ} [MPa]$ ')
plt.show()

# Membrane above area of cross section
u_fem_plot = plt.figure(figsize = (9.8,6.6))
u_fem_plot = ax = Axes3D(u_fem_plot)
u_fem_plot =ax.set_xlabel('X [mm]')
u_fem_plot =ax.set_ylabel('Y [mm]')
u_fem_plot =ax.zaxis.set_rotate_label(False)
u_fem_plot =ax.set_zlabel('$\phi$ [N/mm]',rotation=0)
u_fem_plot = ax.view_init(58,167)
u_fem_plot = ax.plot_trisurf(coordinates[:,0]*scale, coordinates[:,1]*scale, phi,
    triangles=elements, cmap=plt.cm.jet, linewidth = 0.1, edgecolor = 'Black')
plt.show()

##############################################
#PRINTING NUMERIAL RESULTS
##############################################

print('Lenght is:', L,'m')
print('Type of cross section:', name)
print('Area of cross section is :', A*sf,'m^2')
print('Torque is :', T,'Nm')
print('Used solver: ',solver)
print('Number of DOF: ',n_no)
```

```
print('Rate of twist is',theta)
print('Angle of twist is',varphi,'rad')
print ( "Maximum stress from FEM: ",max(napeti)," MPa,")

##############################################
#ANALYTICAL SOLUTION
##############################################
sys.path.append('../analytic')
import analytical as analy

if name == "equilateral triangle":
    Tau_max,I_a = analy.a_sol_eq_tria(Ge,T,a)
    error = abs(I_a-I_n)/I_a*100
    print ('I_n is ',I_n,'and error is ',error,'%')
elif name == "circle":
    Tau_max,I_a = analy.a_sol_cir(T,r)
    error = abs(I_a-I_n)/I_a*100
    print ('I_n is ',I_n,'and error is ',error,'%')

elif name == "right angle triangle":
    Tau_max,theta_a,I_a = analy.a_sol_rat(T,a,Ge)
    error = abs(I_a-I_n)/I_a*100
    print ('I_n is ',I_n,'and error is ',error,'%')
elif name == "rectangle" and el_type == "t":
    Tau_max,theta_a,I_a = analy.a_sol_rect(Ge,T,a,b,L)
    error = abs(I_a-I_n)/I_a*100
    print ('I_n is ',I_n,'and error is ',error,'%')

else:
    print ("analytical solution is not implemented yet")
```

Listing 5.11: FEM4 Main code

```
from __future__ import print_function
from __future__ import division
from __future__ import absolute_import
from __future__ import unicode_literals
import numpy as np
import pylab as plt
import sys
from scipy import sparse
from mpl_toolkits.mplot3d import Axes3D
################################################################################
"""INPUTS"""
################################################################################

#MESH
sys.path.append('../mesh')
import rectangle_quadri as msh
nx = 16
ny = nx
```

```
Lx = .1
Ly = Lx

coordinates, elements,n_no,isOnEdge,el_type = msh.mesh(nx,ny,Lx,Ly)
n_e = elements.shape[0]

#MATERIAL
E = 2.1e11            #Young's modulus of elasticity [Pa]
mu = 0.3              #Poisson ratio
Ge = E/(2*(1+mu))     #Shear modulus [Pa]
D = np.identity(2)*1/Ge #Material matrix

#LOADING
#Symmetry factor = sf
theta1 = 1e-4  #Rate of twist - arbitrary value [rad/m]
T = 1e4  #Torque [Nm]

#Length
L = 1  #[m]

#GAUSS INTEGRATION
n = 2 #number of gauss integration points
sys.path.append('../integration')
import gauss2D as GI
ksi,eta,w = GI.gaussPoints2D(n)

#transformation function #coordinates in local domain transformed to global domain
def XY(ksi,eta):
    global x1, x2,x3,x4,y1,y2,y3,y4
    x = 0.25*(x1*(ksi-1)*(eta-1)-x2*(ksi+1)*(eta-1)+x3*(ksi+1)*(eta+1)-x4*(ksi-1)*(
        eta+1))
    y = 0.25*(y1*(ksi-1)*(eta-1)-y2*(ksi+1)*(eta-1)+y3*(ksi+1)*(eta+1)-y4*(ksi-1)*(
        eta+1))
    return x,y


#Global right hand side
f = np.zeros((n_no,1))
#Sparse matrix - zeros
I = np.zeros((16*n_e),dtype = np.int)
J = np.zeros((16*n_e),dtype = np.int)
V = np.zeros((16*n_e),dtype = np.float64)
#counter
cnt = 0
B_glob= np.zeros((2*n_e,4))
#Area of each trianglular element
A_e = np.zeros(n_e)
#Total area of the cross section
A = 0
###################### Stiffness matrix ####################
for ind,val in enumerate(elements):
#    print (val)
```

```
    X = coordinates[val,:][:,0]
    Y = coordinates[val,:][:,1]
    coord = np.concatenate((X,Y))
    f_e = 0
    K_el = 0
    detsum =0
    for j in range(n**2):
        ##Elements of shape function
        ##Shape function
        N = np.array([[0.25*(ksi[j]-1)*(eta[j]-1),-0.25*(ksi[j]+1)*(eta[j]-1)
            ,0.25*(ksi[j]+1)*(eta[j]+1) ,-0.25*(ksi[j]-1)*(eta[j]+1)]])
        ##derivatives of shape function - local domain (dksi,deta)
        dN = 0.25*np.array([[eta[j]-1,1-eta[j],eta[j]+1,-eta[j]-1],[ksi[j]-1,-1-ksi
            [j],ksi[j]+1,1-ksi[j]]])
        #Jacobian
        Jac = np.array([[np.dot(dN[0,:],np.transpose(X)),np.dot(dN[1,:],np.
            transpose(X))],[np.dot(dN[0,:],np.transpose(Y)),np.dot(dN[1,:],np.
            transpose(Y))]])
        #transpose Jacobian
        Jact = np.transpose(Jac)
        #determinant of Jacobian
        detJ = np.linalg.det(Jac)
        #sum of determinant for each point
        detsum+=detJ*w[j]
        #vector of derivatives of shape functions #derivatives of shape function -
            global domain
        B = np.dot(np.linalg.inv(Jact),dN)
        #transpose vector of derivatives of shape functions
        Bt = np.transpose(B)
        #Local stiffness matrix
        K_el += np.dot(np.dot(Bt,D),B)*detJ*w[j]
        #Local load vector
        f_e += np.transpose(N)*detJ*w[j]
    #Area of element
    A_e[ind] = detsum
    #Total area of the cross section
    A +=detsum
    f[elements[ind,:]]+=f_e*2*theta1
    B_glob[2*int(cnt/16):2*int(cnt/16)+2,:] = B
    #Sparse matrix notation
    I[cnt:16+cnt]+=np.concatenate((val,val,val,val),axis=0)
    J[cnt:16+cnt]+=np.repeat(val,4)
    V[cnt:16+cnt]+=np.ravel(K_el,16)
    cnt+=16
K_sparse_ = sparse.csc_matrix((V,(I,J)),shape=(n_no,n_no))

def funBC(I,J,V,isOnEdge,f):
    for i in plt.find(isOnEdge):
        cnt = 0
        for j in range(len(I)):
            if i ==I[j] or i ==J[j]:
                if I[cnt]!=J[cnt]:
```

```
                    V[cnt]=0
            cnt+=1
    n= np.count_nonzero(V)
    I_n = np.zeros((n),dtype = np.int)
    J_n = np.zeros((n),dtype = np.int)
    V_n = np.zeros(n)
    #INDECES OF NONZEROS ELEMENTS IN V VECTOR
    chs = np.asarray(np.nonzero(V))
    #NEW VECTORS
    V_n = np.take(V,chs).reshape(np.shape(chs)[1]) #Nonzero Values
    I_n = np.take(I,chs).reshape(np.shape(chs)[1]) #Nonzero indexes of rows
    J_n = np.take(J,chs).reshape(np.shape(chs)[1]) #Nonzero indexes of colums

    K_sparse = sparse.csc_matrix((V_n,(I_n,J_n)),shape=(n_no,n_no)).tocsc()
    cnt = 0
    for i in isOnEdge:
        if i==True:
            K_sparse[cnt,cnt]=1.
            f[cnt] = 0.
        cnt+=1
    return K_sparse, f

print ("BCs are done")
K_sparse,f_sparse = funBC(I,J,V,isOnEdge,f)
f_sparse1 = f_sparse[:,0]


sys.path.append('../solver')
import cgm as cgm
phi_q= cgm.my_cg(K_sparse,f_sparse1)

#############################################################################
"""POST PROCESSING"""
#############################################################################
Iu_q=0
T1_q = 0
for i in range(n_e):
    T1_q +=2*A_e[i]/4*np.dot(np.array([1,1,1,1]),phi_q[elements[i,:]]) #Torque
        caused the rate of twist theta1
    ie = elements[i,:]
    Iu_q += np.sum(phi_q[ie])/4.*A_e[i]

print('QUAD T1 is: ',T1_q)
phi_q=phi_q*T/T1_q #Displacement of nodes

Iu_qunit = Iu_q/(2*Ge*theta1)
I_n_q = 4*Iu_qunit #Numerical value of the polar moment of inertia
print('Vu ',I_n_q,' and error ',error)
print('QUAD max phi is ',max(phi_q))

#PLOT MEMBRANE ABOVE THE CROSS-SECTON
phi_plot = plt.figure(figsize=(9.8,6.6))
```

```
phi_plot = ax = Axes3D(phi_plot)
phi_plot = ax.plot_trisurf(coordinates[:,0], coordinates[:,1], phi_q, cmap=plt.cm.
    jet, linewidth = 0.1, edgecolor = 'Black')
phi_plot =ax.set_xlabel('X [m]')
phi_plot =ax.set_ylabel('Y [m]')
phi_plot = ax.set_title('FEM quadrilateral elements potential')
phi_plot =ax.zaxis.set_rotate_label(False)
phi_plot =ax.set_zlabel('$\phi$',size = 20,rotation=180)
phi_plot = ax.view_init(51,14)
plt.show()

#CALL FUNCTION TO STRESS CALCULATION
sys.path.append('../stress')
import stress as stres
Stress, coord,Average = stres.stress(phi_q,D,elements,coordinates)

Average/=1e6 #Stress from Pa to MPa

#CALL FUNCTION TO TRANSFORM CODE TO PARAVIEW NOTATION
sys.path.append('../paraview')
import output_to_vtk_paraview as wVTK
wVTK.WriteVTK('plot1',coordinates, elements, Average)
```

Gaussian quadrature

Listing 5.12: Gaussian integration

```
#The function generates Gauss points (1,4,9) for 2D integration
def gaussPoints2D(n):
    if (n==1): #one integration point
        ksi =eta = [0.]
        w = [4.]
    elif (n==2): #four integration points
        ksi = (-0.5773502691896,0.5773502691896,-0.5773502691896,0.5773502691896)
        eta = (-0.5773502691896,0.5773502691896,0.5773502691896,-0.5773502691896)
        w =  (1.,1.,1.,1.)
    elif (n==3): #nine integration points per element
        ksi =
            (-0.774596669241483,0,0.774596669241483,-0.7745966692414830,0,0.774596669241483,-0.7

        eta =
            (-0.774596669241483,-0.774596669241483,-0.7745966692414830,0,0,0,0.774596669241483,0

        w =  (25/81,40/81,25/81,40/81,64/81,40/81,25/81,40/81,25/81)
    else:
        print ('not implemented yet')
        ksi = eta = w=np.NaN
    return ksi,eta,w
```

Quadrilateral stress

Listing 5.13: Stress FEM4

```
import numpy as np
#This function evaluates stresses in Gauss integration points
def stress(phi,D,elements,coordinates):
    cnt = 0
    n_e = elements.shape[0]
    Stress = np.zeros((n_e*4,4))
    coord = np.zeros((n_e*4,2))
    Average = np.zeros(n_e)
    ksi = (-0.5773502691896,0.5773502691896,0.5773502691896,-0.5773502691896)
    eta = (-0.5773502691896,-0.5773502691896,0.5773502691896,0.5773502691896)
    w =  (1.,1.,1.,1.)
    for ind,val in enumerate(elements):
        X = coordinates[val,:][:,0]
        Y = coordinates[val,:][:,1]
        phi_i = phi[val]
        for j in range(4):
            #Elements of shape function
            x = 0.25*(X[0]*(ksi[j]-1)*(eta[j]-1)-X[1]*(ksi[j]+1)*(eta[j]-1)+X[2]*(
                ksi[j]+1)*(eta[j]+1)-X[3]*(ksi[j]-1)*(eta[j]+1))
            y = 0.25*(Y[0]*(ksi[j]-1)*(eta[j]-1)-Y[1]*(ksi[j]+1)*(eta[j]-1)+Y[2]*(
                ksi[j]+1)*(eta[j]+1)-Y[3]*(ksi[j]-1)*(eta[j]+1))
            #derivatives of shape function - local domain (dksi,deta)
            dN = 0.25*np.array([[eta[j]-1,1-eta[j],eta[j]+1,-eta[j]-1],[ksi[j]-1,-1-
                ksi[j],ksi[j]+1,1-ksi[j]]])
            #Jacobian
            Jac = np.array([[np.dot(dN[0,:],np.transpose(X)),np.dot(dN[1,:],np.
                transpose(X))],[np.dot(dN[0,:],np.transpose(Y)),np.dot(dN[1,:],np.
                transpose(Y))]])
            #transpose Jacobian
            Jact = np.transpose(Jac)
            #vector of derivatives of shape functions #derivatives of shape function
                 - global domain
            B = np.dot(np.linalg.inv(Jact),dN)
            Stress_ = np.dot(B,phi_i)
            Stress[cnt,:]=val[j],Stress_[0],Stress_[1],np.sqrt(Stress_[0]**2+Stress_
                [1]**2)
            Average[ind] +=Stress[cnt,3]
            coord[cnt,:]=x,y
            cnt+=1
    Average/=4
    return Stress,coord,Average
```

Paraview notation

Listing 5.14: Paraview transformation

```
import numpy as np
# This function transforms code to Paraview friendly. Software Paraview visualizes
     the stress on quadrilateral elements.
def WriteVTK(tittle,coordinates,elements,stress):
    cell_ty = 9   #cell type
    n_el = elements.shape[0] #number of elements
    #Write the array to tittle.vtk
    with file(tittle+'.vtk', 'w') as vtkfile:
        vtkfile.write('# vtk DataFile Version 3.0 \n')
        vtkfile.write('2D Unstructured Grid of Linear Quadrilaterals \n')
        vtkfile.write('ASCII\n')
        vtkfile.write('\n')
        vtkfile.write('DATASET UNSTRUCTURED_GRID\n')
        vtkfile.write('POINTS {0} float\n'.format(coordinates.shape[0]))

        for i in coordinates:
            x = np.array([i[0],i[1],0])
            vtkfile.write(" ".join(map(str, x)))
            vtkfile.write('\n')
        vtkfile.write('\n')
        vtkfile.write('CELLS {0} {1}\n'.format(n_el,5*n_el))
        for i in elements:
            y = np.array([4])
            x = np.concatenate((y,i),axis = 1)
            vtkfile.write(" ".join(map(str, x)))
            vtkfile.write('\n')
        vtkfile.write('\n')
        vtkfile.write('CELL_TYPES {0}\n'.format(n_el))
        for i in range(n_el):
            vtkfile.write('{0}\n'.format(cell_ty))
        vtkfile.write('\n')
        vtkfile.write('CELL_DATA {0}\n'.format(n_el))
        vtkfile.write('SCALARS {0} float\n'.format('Stress[MPa]'))
        vtkfile.write('LOOKUP_TABLE default\n')
        for i in stress:
            vtkfile.write('{0}\n'.format(i))
        vtkfile.close()
```

External mesh generator

Listing 5.15: External mesh generator

```
import numpy as np
#The function can transform triangular mesh from MSC notation to TorPy notation
def mesh(fileName):
    cntUnsortNodes = 0
    stop = False
    g2l = {}
    cntElem = 0
    with open(fileName) as infile:
        for i,line in enumerate(infile):
            if (stop):break
            if (i==2):   #number of nodes and elements
                for j,iline in enumerate(line.split()):
                    if (j==4):
                        nNodes=int(iline)
                        coordinates = np.zeros((nNodes,3),dtype=np.float)
                        unsorteNodesNumber = np.zeros(nNodes,dtype=np.int32)
                    if (j==5):
                        nElements=int(iline)
                        elements = np.zeros((nElements,3),dtype=np.int32)
            if (i>3):
                if ((i-1)%3==0):# first from triple
                    for j,iline in enumerate(line.split()):
                        if (j==1):
                            unsorteNodesNumber[cntUnsortNodes] = iline
                            g2l[np.int32(iline)-1]=cntUnsortNodes
                            cntUnsortNodes += 1
                if ((i-2)%3==0):# second from triple
                    for j,iline in enumerate(line.split()):
                        coordinates[cntUnsortNodes-1,j]=iline
                        if (cntUnsortNodes>(nNodes-1)):
                            stop=True
        lineNumberCoordinates = i;
        stop = False
        for i,line in enumerate(infile):
            if (stop):break
            if (1):
                if ((i)%3==0):
                    tmp = line.split()
                if ((i+1)%3==0):# third from triple
                    for j,iline in enumerate(line.split()):
                        elements[cntElem,j]=g2l[np.int32(iline)-1]
                    cntElem+=1
                    if (cntElem==nElements):
                        stop = True;
    coord = np.zeros((nNodes,2),dtype=np.float) #2D triangular mesh
    coord = coordinates[:,0:2]
    return coord,elements
```

E BEM Codes

Listing 5.16: BEM Main code

```
#IMPORT FUNCTIONS
from __future__ import print_function
from __future__ import division
import numpy as np
import pylab as plt
import gauss
from mpl_toolkits.mplot3d import Axes3D
import sys
################################################################################
"""INPUTS"""
################################################################################
#MATERIAL
E = 2.1e11            #Young's modulus of elasticity [Pa]
mu = 0.3              #Poisson ratio
Ge = E/(2*(1+mu))     #Shear modulus [Pa]

#LOADING
theta = 1e-4  #rate of twist [rad/m]
#NUMERICAL INTEGRATION
NOP =2 #number of points in Gauss quadrature
omega,w,err = gauss.gaussPoints_1D(NOP) #import Gauss points: interval <-1,1>
omega = list(omega) # Gauss points: remaping from interval <-1,1> to interval
    <0,1>
w   = list(w)
for i in range(len(omega)):
   omega[i] = 0.5*(omega[i] + 1)
   w[i] *= 0.5

#MESH
mesh = 'experiment' # choose mesh - manual, experiment or patran

if mesh == 'manual':     #import mesh from mesh generator
    sys.path.append('../..//FEM/mesh')
    name = 'circle'  # choose circle, rectangle, equilateral triangle or right
        angle triangle
    import select_mesh as msh
    coordinates, elements,n_no,isOnEdge,el_type,sf,dim = msh.mesh(name) #import
        mesh from FEM generator
    import mesh_fe_to_be_new as mbefe
    coor_nod,nEl,bound_final= mbefe.femTObem(coordinates, elements) # modification
        FEM mesh to BEM mesh

if mesh == 'patran':
    sys.path.append('../..//PATRAN_EXPORT_MESH')
    import patran_reader_func2D as msh
    name = 'hole.out'           #choose name of the exported mesh from Patran or
        Marc
```

```
43      coordinates, elements= msh.mesh(name)
44      import mesh_fe_to_be_new as mbefe
45      coor_nod,nEl,bound_final= mbefe.femTObem(coordinates, elements) # modification
            FEM mesh to BEM mesh
46
47  if mesh == 'experiment':
48      import mesh_example as msh
49      coordinates,bound_final,coor_nod,nEl,elements,inner= msh.mesh() #manual mesh
50      name = 'exper'
51
52  if name=='circle':
53      sf = 4
54  else:
55      sf = 1
56
57  ###############################################################
58  #INPUT must contain variables: bound_final,coordinates, coor_nod, nEl,elements,BCu
59  ###############################################################
60
61  ###############################################################################
62  """SOLUTION"""
63  ###############################################################################
64
65  #CALCULATION OF NORMALS
66  edge = np.unique(np.concatenate(bound_final)) #outer nodes of FEM
67  tang = coordinates[bound_final[:,1],:] - coordinates[bound_final[:,0],:]
68  n_t = np.sqrt(tang[:,0]**2+tang[:,1]**2) #lenght of tangent
69  tang[:,0] /= n_t #unit tangents of edges
70  tang[:,1] /= n_t #unit tangents of edges
71  normals = np.vstack((tang[:,1] , -tang[:,0])) #normals of edges
72
73  #INNER and OUTER nodes
74  all_nodes = np.arange(0,coordinates.shape[0]) #all nodes
75  inner = np.setdiff1d(all_nodes,edge) #number of inner nodes
76
77  #ZEROS GLOBAL MATRICES OF COEFFICIENTS
78  H = np.zeros((nEl,nEl),dtype = np.float64)
79  G = np.zeros((nEl,nEl),dtype = np.float64)
80
81  #BOUNDARY CONDITIONS
82  BCu = np.zeros(nEl,dtype=np.bool) #zero matrix
83  if name=='circle':
84      for i in range(int(nEl/3)):
85          BCu[i+nEl/3]=1
86  elif name == 'exper':
87      BCu=np.array([False,True,True,False])
88  else:
89      BCu = np.ones(bound_final.shape[0])
90
91  ###############################################################################
92  ##SOLUTION ON BOUNDARIES
93  ###############################################################################
```

```
BC = np.zeros(nEl,dtype = np.float64)

#BOUNDARY CONDITIONS FIELD VARIABLE CALCULATION
for i in range(nEl):
  xy_i = coor_nod[i,:]
  if BCu[i] == True:
      BC[i] = .25*(coor_nod[i,0]**2+coor_nod[i,1]**2)

  for j in range(nEl):
    el_j = bound_final[j,:]
    xy_j = coordinates[el_j,:]
    #NON-DIAGONALS ELEMENTS
    if i != j:
      a = xy_j[0,:]
      b = xy_j[1,:]
      len_ab = np.linalg.norm(a-b)
      G_ij = 0
      H_ij = 0
      for m in range(0,NOP):         # numerical integration - vectors r and x
          xx = a + (b-a)*omega[m]
          rr=xx-xy_i
          G_ij += w[m]*np.log(np.linalg.norm(rr))*len_ab #G - element matrix
          H_ij += w[m]*(np.dot(rr,normals[:,j]))/np.linalg.norm(rr)**2*len_ab #H -
              element matrix
      G_ij /= -2*np.pi
      H_ij /= -2*np.pi
      H[i,j] = H_ij
      G[i,j] = G_ij
    #DIAGONALS ELEMENTS
    else:
      a = xy_j[0,:]
      b = xy_j[1,:]
      len_ab = np.linalg.norm(a-b)
      H[i,j]=0.5
      # DIAGONAL ELEMENTS OF G ARE INTEGRATED ANALYTICALLY
      G[i,j] = 0.5/np.pi*np.linalg.norm(len_ab)*(1+np.log(2) - np.log(np.linalg.
          norm(len_ab)))

#APPLICATION OF BC AND CALCULATION OF SYSTEM OF EQUATIONS
dimen = H.shape[0]
HG = np.concatenate((H, G), axis=1)

#FUNCTION FOR COLUMN SWAPPING
def sw_col(arr, frm, to):
    arr[:,[frm, to]] = -arr[:,[to, frm]]
for inx,val in enumerate(BCu):
    if val==True:
        sw_col(HG, inx, inx+dimen)

A,B= np.hsplit(HG, 2)

f = np.dot(B,BC)   #right-hand side
```

```
144
145 #SOLUTION OF SYSTEM Ax = f
146 unk = np.linalg.solve(A,f) #solution
147 u_outer = np.zeros(dimen)
148 q_outer = np.zeros(dimen)
149
150 #GLOBAL VECTORS u and q
151 for i,val in enumerate(BCu):
152     if val==True:
153         u_outer[i]=BC[i]
154         q_outer[i]=unk[i]
155     else:
156         u_outer[i]=unk[i]
157         q_outer[i]=BC[i]
158
159 ##############################################################################
160 ##SOLUTION OF INNER NODES
161 ##############################################################################
162
163 #NEW VECTORS- INTEGRATION OF INNER NODES
164 G_ = np.zeros([nEl])
165 H_ = np.zeros([nEl])
166
167 #POTENTIAL OF INNER NODES
168 u=np.zeros(coordinates.shape[0])
169
170 #LAPLACE TO POISSON
171 u_check = u_outer - 0.25*(coor_nod[:,0]**2+coor_nod[:,1]**2)
172 for i,item in enumerate(inner):
173     xi = coordinates[item,:] #coordinates of inner nodes
174 #    print(coordinates[item,:])
175     for j in range(bound_final.shape[0]):
176         el_j = bound_final[j,:]
177         xy_j = coordinates[el_j,:]
178         a = xy_j[0,:]
179         b = xy_j[1,:]
180         len_ab = np.linalg.norm(a-b)
181         H_inn = 0
182         G_inn = 0
183         for m in range(0,NOP):
184             xx = a + (b-a)*omega[m]
185             rr=xx-xi
186             G_inn += w[m]*np.log(np.linalg.norm(rr))*len_ab
187             H_inn += w[m]*(np.dot(rr,normals[:,j]))/(np.linalg.norm(rr))**2*len_ab
188         H_inn /= -2*np.pi
189         G_inn /= -2*np.pi
190         H_[j]=H_inn
191         G_[j]=G_inn
192     u[item] = (np.dot(G_,q_outer))-(np.dot(H_,u_outer))-0.25*(xi[0]**2+xi[1]**2)
193
194 for i,item in enumerate(bound_final[:,0]):
195     u[item]=u_check[i]
```

```
196 phi_bem = 2*u*Ge*theta/1000
197
198 Area_h_el = np.zeros(elements.shape[0],dtype=np.float64)
199 Area_w = 0.0   #whole area of Cross-Section
200 Iu = 0.0
201 if mesh == 'manual':
202     for i in range(elements.shape[0]):
203         ie = elements[i,:] #numbers of i-th elements in tmp variable
204         x = coordinates[ie,0]
205         y = coordinates[ie,1]
206         Ah = np.array([[1.0,x[0],y[0]],[1.0,x[1],y[1]],[1.0,x[2],y[2]]])
207         Area_h = np.linalg.det(Ah)*0.5 #Ghx = dMhx*iAh
208         Area_h_el[i] = Area_h
209         Area_w +=Area_h
210         ie = elements[i,:]
211         np.sum(u[ie])
212         Iu += np.sum(u[ie])/3.*Area_h_el[i]
213     print('Volume under membrane is ',Iu)
214     I_n = sf*4*Iu
215     print ('I_n',I_n)
216 #    error = abs(I_a-I_n)/I_a*100
217 #    print ('I_n BEM is ',I_n,'and error is ',error,'%')
218
219
220 ###############################################################################
221 """POST PROCESSING"""
222 ###############################################################################
223 #plot the BE mesh
224 plt.figure()
225 plt.gca().set_aspect('equal')
226 plt.plot(coor_nod[:,0],coor_nod[:,1],'bo')
227 plt.grid(True)
228 plt.show()
229
230 #plot the BE membrane
231 u_bem_plot = plt.figure()
232 u_bem_plot = ax = Axes3D(u_bem_plot)
233 u_bem_plot = ax.plot_trisurf(coordinates[:,0], coordinates[:,1], phi_bem, cmap=plt
        .cm.jet, linewidth = 0.1, edgecolor = 'Black')
234 u_bem_plot = plt.show()
```

Listing 5.17: Mesh FEM to BEM

```
# -*- coding: utf-8 -*-
import numpy as np
#This function can transfrom FE mesh to BE mesh. The generated mesh uses constant
    element discretisation

#################MODIFY FEM MESH TO BEM MESH
def femTObem(coordinates, elements):
    bound = np.zeros([3*elements.shape[0],2],dtype=np.int)
    for i in range(elements.shape[0]):
        bound[i,:]=elements[i,0:2]
        bound[i+elements.shape[0],:]=elements[i,1:3]
        bound[i+2*elements.shape[0],:]=elements[i,0:3:2]
        if bound[i,0]>bound[i,1]:
            temp=bound[i,0]
            bound[i,0]=bound[i,1]
            bound[i,1]=temp
        if bound[i+elements.shape[0],0]>bound[i+elements.shape[0],1]:
            temp=bound[i+elements.shape[0],0]
            bound[i+elements.shape[0],0]=bound[i+elements.shape[0],1]
            bound[i+elements.shape[0],1]=temp
        if bound[i+2*elements.shape[0],0]>bound[i+2*elements.shape[0],1]:
            temp=bound[i+2*elements.shape[0],0]
            bound[i+2*elements.shape[0],0]=bound[i+2*elements.shape[0],1]
            bound[i+2*elements.shape[0],1]=temp

    pos_del = []
    for i in range(3*elements.shape[0]):
        for j in range(3*elements.shape[0]):
            if bound[i,0]==bound[j,0] and bound[i,1]==bound[j,1]:
                if i!=j:
                    pos_del.append(i)
                    pos_del.append(j)

    bound = np.delete(bound,pos_del,0)
    bound_final = np.zeros([bound.shape[0],2],dtype=np.int)
    first_row = 0  #starting point of integration
    bound_final[0,:]= bound[first_row,:]
    bound = np.delete(bound,first_row,0)
    for i in range(bound.shape[0]):

        if bound[0,0]==bound_final[i,1]:
            bound_final[i+1,:]=bound[0,:]
            bound = np.delete(bound,0,0)

        else:
            for j in range(bound.shape[0]) :
                if bound[j,0]==bound_final[i,1]:
                    bound_final[i+1,:]=bound[j,:]
                    bound = np.delete(bound,j,0)
                    break
```

```
            elif bound[j,1]==bound_final[i,1]:
                bound_final[i+1,:]=bound[j,1],bound[j,0]
                bound = np.delete(bound,j,0)
                break

    nEl = bound_final.shape[0]

    coor_nod = np.zeros([bound_final.shape[0],2],dtype=np.float64)

    #coordinates of BEM nodes (between FEM nodes on boundary)
    for i,item in enumerate(bound_final):
        coor_nod[i,:]=(coordinates[item[1]]+coordinates[item[0]])/2.
    return coor_nod,nEl,bound_final
```